Data Providers' Handbook

Archiving Guide to the PDS4 Data Standards

DRAFT

Data Design Working Group

1 April 2013

Version 0.3.10

CHANGE LOG

Revision	Date	Description	Author
0.1	Mar 30, 2009	Initial draft based on information collected by the Data System Working Group.	R.Joyner
0.2	Aug 6, 2009	Updated versions of all Classes	R. Joyner
0.2.1	2010-08-31	Complete overhaul, but only partly successful through page 25	R. Simpson
0.22	Aug 31, 2010	Integrate Simpson and Joyner docs	R. Joyner etal
0.22.1	2010-09-22	Edits through Section 3, except Section 2	Simpson
0.22.2	2010-09-24	Added comments from Mitch Gordon	Simpson
0.22.2	2010-10-01	Significant edits to Section 4	Simpson
0.22.3	2010-10-25	Significant edits to Sections 1,3-6	Simpson
0.3	2011-04-15	Significant edits addressing Build 1b comments	M.Gordon etal
0.3.1	2011-04-21	Additional content added	M.Gordon etal
0.3.2	2011-05-02	Significant reorganization, additional edits	M.Gordon etal
0.3.3	2011-06-29	Make Sections 1-2 more user friendly	R. Simpson, M. Gordon
0.3.4	2011-09-06	Major changes Sections 2, 3, 8,14	M. Gordon, R. Joyner
0.3.5	2011-10-31	Updates for schema 0.5.00g	M. Gordon, R. Joyner
0.3.6	2012-01-31	Updates for schema 0.7.0.0.j	M. Gordon, R. Joyner
0.3.7	2012-03-28	Updates for schema 0.7.0.0.j	M. Gordon, R. Joyner
0.3.8	2012-05-28	Updates for schema 0.8.0.0.k	M. Gordon, R. Joyner
0.3.9	2012-10-01	Updates for schema 0.3.0.0a	M. Gordon, R. Joyner
0.3.10	2013-04-01	Major changes for schema 0.3.1.0b	M. Gordon, R. Joyner, R. Simpson

TABLE OF CONTENTS

1.0 INTRODUCTION

Planetary Data System version 4 (PDS4) represents a departure from previous versions of the PDS. Although it is still an archive of planetary data, it has been designed using contemporary information technology concepts and tools. The system is built around a 'data model' that rigorously defines each of its components and the relationships among them. There are only four fundamental data structures, but many extensions are possible — each rigorously defined. By carefully controlling product definitions and relationships, PDS can accurately track the progress of each product entering the system, compute detailed inventories of holdings, design sophisticated services that users can request to act on subsets of the archive (such as transformations and displays, in addition to the expected search and retrieval functions), and connect data products to relevant internal and external information (documentation).

Note that this version of the document is conformant to version "0.3.1.0.b" of the Information Model.

1.1 Purpose

The *Data Providers Handbook* (*DPH*) is a guide for preparation of data being submitted to PDS4. It will walk you through preparation of very simple products, collections of products, and bundles of collections, which are the units in which deliveries are made to PDS4.

1.2 Audience

The *DPH* is written for scientists and engineers in the planetary science community who are planning to submit new or restored data to PDS4 (data providers)[1]. While the document is applicable to all such submissions, most of the examples and discussions are presented in a mission/instrument context.

1.3 Reader Preparation

The *DPH* is one of several documents[2] describing the PDS4 system.

[1] PDS4 standards are, in general, not backwardly compatible with version 3 (PDS3). In principle, version 4 data can be described using version 3 labels, but the converse is not true; some PDS3 structures are no longer supported under PDS4.

[2] The XML schema document (XSD), the schematron (SCH), the IM-specification, and the Standards Reference are "views" into the IM and are collectively the "Standards".

Readers, even those very familiar with previous versions of PDS, **should read the *PDS4 Concepts* document [4] before beginning the *Data Provider's Handbook.***

The *DPH* should be used in conjunction with the *PDS Standards Reference* [2], the PDS4_PDS schema and Schematron files, and the *PDS4 Data Dictionary* [3a,b], which collectively provide the information necessary to develop a PDS4 compliant archive.

1.4 XML Editors

PDS4 is implemented in the eXtensible Markup Language (XML), a set of 'open source' rules for encoding documents and data structures in machine-readable form with special applicability to providing web services. It is beyond the scope of the *DPH* to include an XML tutorial. Consult external sources for information on XML.

The use of an XML editor simplifies construction and validation of schemas (the plural of 'schema') and labels. Two editors have been popular during development of PDS4.

oXygen **http://www.oxygenxml.com** which is licensed software, and
Eclipse **http://www.eclipse.org/** which is open source software.

1.5 Applicable Documents

1.5.1 Controlling Document

[1] Planetary Data System (PDS) PDS4 Information Model Specification, Version 0.3.1.0.b

1.5.2 Reference Documents

[2] Planetary Data System Standards Reference, v.4.0.8, April 2013.

[3a] PDS4 Data Dictionary - Abridged - V.0.3.1.0.b.

[3b] PDS4 Data Dictionary - Unabridged - V.0.3.1.0.b.

[4] *PDS4 Concepts*, April 2013.

1.5.3 Document Availability

PDS4 documents governing archive preparation are available online:

http://pds.nasa.gov/pds4/doc/

For questions concerning these documents, contact any PDS data engineer or contact the PDS Operator at pds_operator@jpl.nasa.gov or 818-393-7165.

1.6 Other Resources

1.6.1 XML Schema location

http://pds.nasa.gov/pds4/schema/released/pds/

1.6.2 PDS4 Software

http://pds.nasa.gov/pds4/software/

1.6.3 PDS4 Wiki

- The PDS Wiki is a primer for 'How to Create a PDS4 Label' and can be found at:

  https://oodt.jpl.nasa.gov/wiki/display/pdscollaboration/How+to+Create+a+PDS4+Label+-+FOR+REVIEW

1.6.4 SBN PDS4 Data Migration Wiki

- This site contains information about the PDS4 prototype migration effort at the Small Bodies Node. It will eventually contain both tracking information and potentially useful support information for PDS4 data developers inside and outside the SBN.

  http://borrelly.astro.umd.edu/wiki/SBN_PDS4_Migration_Wiki

PART I. PRELIMINARIES

2.0 BUILDING BLOCKS

2.1 Terminology

The results of our exploration of the Solar System can be loosely described as *data objects*; these can be electronic files, dust samples, or a sense of awe at the wonder of the universe. For purposes of archiving, we need a *description* to accompany each data object — in the case of an electronic file, a digital object, we need both the structure and meaning of the file contents for it to be useful. We can't fit dust samples or senses of awe into PDS4, but we can fit their descriptions. A description paired with its data object (when available — *e.g.*, if a digital object) is called an *information object*. If many data objects have similar characteristics, we can group them into a *class* and the common characteristics are the defining *attributes* of that class.

A *product* is one or more closely related information objects for which the descriptions have been combined into a single XML *label* and for which the product has a PDS-unique *logical identifier*. Closely related products may be grouped into a *collection*; in fact, every product entering PDS must be a member of some collection. Closely related collections may be grouped in a *bundle*.

For example, a planetary image, the histogram of its pixel values, and the descriptions of both could be organized as a product. Many such products — perhaps of the same target — could be defined as a collection. Image collections from many targets along with appropriate documentation, calibration, etc. (separate collections) could be a bundle, which would be a deliverable to PDS.

A few words have meanings which differ depending on the community in which they are used. We have adopted modifiers to help distinguish among multiple uses. For example, 'attribute' is widely used in both PDS and XML — but its meaning in each case is different. In this document we use 'attribute' and 'XML attribute' to establish the context.

For more rigorous definitions of the terms introduced above, see the *PDS4 Glossary,* Appendix A in the *PDS4 Concepts* document.

2.2 Data

A word of caution about terminology – we have tried to avoid using differently terms that have strong PDS3 connotations. Unfortunately the English language does not provide a sufficient set of meaningful, unique, unambiguous terms to meet all of our needs. Please do not rely on the names of things – review carefully the PDS4 definitions.

Four basic structural data formats are allowed in PDS4.

1. homogeneous_array_structure
    - Suitable for images, spectra, spectral cubes, maps, etc.
    - The elements of an array are homogeneous.
    - The individual elements of any array are stored with their bytes in the order dictated by their scalar type.
    - PDS requires a specific order in which the elements of the array are stored. The order is described in the *Standards Reference [2], Section 4A*.

The majority of PDS4 objects can be supported by these two structures. For those PDS4 objects which cannot be supported by the above, there are two additional structures distinguished by whether or not software must be used to decode the information before it can be accessed for reading, display, or analysis.

2. repeating_record_structure
    - Suitable for fixed length tabular data.
    - May be either binary or character, but a single object must be defined as one or the other (not mixed).
    - The records are fixed length.
    - The fields of a record may be heterogeneous.
    - The formats (data type and size) of fields in the same position in each record are the same (i.e., the second field in the second record is constructed identically to the second field in the first record).

3. parsable_structure
    - Suitable for plain text, HTML, XML, tabular data with variable length fields and records (delimited text).
    - The contents are a byte stream which can be parsed with standard rules (e.g., comma separated entries, standard punctuation); no decoding software (e.g., Adobe Acrobat©) is required.
    - The attribute parsing_standard_id is used to identify the parsing standard to be used.

4. encoded_structure
    - Suitable for documents, browse products, etc., but generally not for observational products.
    - Contents are a byte stream that must be decoded by software before use.
    - The use of encoded_byte_stream objects is restricted by PDS to a limited set of PDS approved external standards (e.g., PDF/A, JPEG, GIF).
    - Only in exceptional cases will encoded_byte_stream objects be considered appropriate for storing observational data. Prior PDS approval is required.

Each of these structural formats corresponds to one PDS4 “base class” and each PDS4 “base class” uses one and only one of the structural formats.

Structural Format	Base Class

- repeating_record_structure ↔ Table_Base
- homogeneous_array_structure ↔ Array_Base
- parsable_structure ↔ Parsable_Byte_Stream
- encoded_structure ↔ Encoded_Byte_Stream

Here are a few rules (for a full description, see the *Standards Reference [2], Section 4*):

- Each digital object must be stored in one of the four basic data formats.
- A digital object must be contained in a single file (i.e., a digital object cannot span multiple files)
- A file may contain multiple digital objects.
- Digital objects within a file are not required to use the same storage structure.
- When multiple digital objects are contained in a single file, each must be contiguous (they may not overlap in storage)

2.2.1 Sample Data

This document frequently references a set of example PDS4 products – see Section 14 - Example PDS4 Products.

There are two different sets of examples:

- A set of example products. This includes a representative set of products that would exist within an archive (e.g., character table, binary table, document, etc.).

- An example of a complete archive — a bundle with collections, each having products.

3.0 SCHEMA AND LABELS – AN OVERVIEW

Label development begins with an XML schema. PDS maintains a master-schema that serves as a library of generic XML schema for each PDS4 product type.

In order for any XML document (including a PDS label) to meet the XML standard, it must be both "well formed" and "valid". A well-formed XML document must have correct XML syntax; a valid XML document must conform to the rules of an XML schema document (XSD) and, if applicable, an XML Schematron (SCH) document. Under PDS4, schemas provide the rules governing the structure and some of the content of each XML class, and Schematron documents further constrain the contents of those classes.

In consultation with your PDS discipline node (DN), for each product type, determine which product labels you will need. The DN may assist you in developing individual labels by producing template XML files from the master-schema, possibly incorporating both discipline and mission specific areas if appropriate for the products you plan to produce. Alternatively, they may advise you as you develop XML 'instances' from the master-schema.

As the data provider, you may produce a mission or instrument specific dictionary in which you define additional classes and attributes specific to the data you will produce. Such additional classes and attributes may be placed in the mission area of your XML labels. You might prepare an additional Schematron file containing "rules" that constrain or restrict values for a particular attribute.

Finished labels must be validated against the master-schema, the master-Schematron, and any specific discipline and mission specific schema and Schematron on which they are based. Thus, in order for a label to be compliant with PDS4 standards, the label must:

- Have correct XML syntax
- Be compliant with the class and attribute structures defined by the PDS master-schema and any relevant discipline node and mission dictionary schemas (XSDs).
- Be compliant with the rules governing specific attributes and their values as set by the PDS Schematron schemas and any relevant discipline node and mission Schematron schemas.

3.1 Pipeline Considerations

If you are developing a collection with more than a few products, you will want to automate label generation as much as possible.

There are at least two approaches:

a) Use a schema (xsd file) as input to the pipeline software you use to generate your labels, or

b) Use a label-template (xml file) as input to the pipeline software. One feature of many XML editors is the ability to generate such a template.

The template looks like the final label except that, depending on how you set some options, there are either no values between the XML tags, or the values which vary from one label to the next are represented by placeholders which the pipeline software will replace.

As with everything, there are advantages and disadvantages to each approach. The first consideration will probably be the software package underlying your pipeline and whether or not it is specifically designed to handle XML (e.g., the PDS supplied Label Generation Tool).

In this handbook, we assume the pipeline will use an XML template. This provides a framework for the discussion that follows. We use Product_Table_Character for most of our examples. The tailored XML label template you receive from your node may differ slightly from the ones in these examples. Our goal here is to become familiar with the contents and to understand the process.

3.1.1 PDS4 Migration Considerations

The SBN PDS4 Migration Wiki contains information about migrating PDS4 products. The site also provides useful support information for PDS4 data developers inside and outside the SBN.

http://borrelly.astro.umd.edu/wiki/SBN_PDS4_Migration_Wiki

3.1.2 Label Generation Tool

The Generate Tool provides a command-line interface for generating PDS4 Labels from either a PDS3 Label or a PDS-specific DOM object. The Generate Tool provides the ability to automatically generate values that do not map directly to the input data object (i.e. PDS3 label).

http://pds.nasa.gov/pds4/software/generate/

3.2 Permitted Schema Modifications

The PDS4 master-schemas (XSD and SCH) are supplied to data providers by the PDS. Missions and other data providers may not modify these existing schemas; however, they may extend existing classes and provide their own additional attributes in their own dictionary schemas (XSD and SCH), with the approval of a PDS discipline node. These schemas must still satisfy the parent schema's restrictions. In principle this means you can modify the schema to make it more restrictive, and you may also add classes within certain constraints.

The PDS4 supplied master-schema must not be modified. This schema is to be considered "read-only" and the content is immutable. If you think you have found a reason to modify the master-schema, contact your discipline node as something has gone terribly wrong.

The discipline and mission schemas are subject to modification. Your discipline node will make a best effort attempt to create 'discipline' and 'mission' schemas appropriate for the products you plan to produce. But, as stated above, you might find that you are able to make modifications that will more closely fit the particular nature of the observational and supplementary products in the archive.

The following is a summary of the type of modifications that you, as the data provider, might find necessary to make:

a) You may change an XML element from optional (minOccurs="0") to required (minOccurs="1", or some number that is less than or equal to the value specified in maxOccurs).
b) You may delete an optional XML element (if minOccurs="0")
c) You may restrict the upper limit on the number of times the XML element will be used by setting maxOccurs to a value that is greater than or equal to the value specified in minOccurs, but less than or equal to the value of maxOccurs in the master-schema.
d) For those XML elements which have "maxOccurs = unbounded", you may set an explicit upper limit on the number of times the XML element will be used.
e) If an XML element will have the same value for all products being produced from this schema you may insert that value into the XML element in the schema.
f) During the initial tailoring, the PDS node staff may insert additional classes in either the Mission_Area and / or the Discipline_Area -- a subsection within the Observation_Area.
g) The data provider may insert additional classes in the Mission Area -- a subsection within the Observation_Area.

You may not modify the minOccurs or maxOccurs attributes such that they would specify a less restrictive set of conditions. For example, if "minOccurs = 5" you cannot set "minOccurs = 4", as this would violate the initial restriction. Similarly, if "maxOccurs = 1", you cannot set "maxOccurs = 2".

There are really three options for what you will do in your schema with XML elements which are 'optional' in the parent schema:

1. The XML element will be used across all labels
2. The XML element will sometimes be used in one or more labels.
3. The XML element will not be used in any labels

There are several approaches to handling optional XML elements based at least in part on your label pipeline design. We recommend:

- Set "minOccurs = 1" (or some appropriate higher value) for the optional XML elements you intend to use in every label.
- Delete the optional XML elements that will not be used in any label (if "minOccurs = 0").
- Leave "minOccurs = 0" for the optional XML elements you intend to use only for some subset of the products. When maxOccurs is unbounded, reset it to an appropriate upper value. Note maxOccurs must be greater than or equal to minOccurs.

Additional constraints can be addressed by creating new and/or modifying existing "rules" in a Schematron (SCH) that is discipline or mission specific. For additional information, consult your DN.

3.3 Developing Local Data Dictionaries

Local data dictionaries (e.g., mission and discipline dictionaries) are developed by the data preparer in conjunction with discussions and review by the lead PDS discipline node. Local dictionaries contain new classes and attributes as needed to provide detailed product descriptions which cannot be defined using the existing PDS classes and attributes

A local dictionary is created when the data preparer needs to define attributes and classes specific to his mission, observing campaign, data restoration effort, etc, including but not limited to specific instrument parameters, and additional observational parameters. They are an essential tool for providing data preparer's latitude to tailor their labels to more closely fit the particular nature of the observational and supplementary data in the archive.

A local dictionary must reside within a namespace that is unique across all other locally defined dictionaries. To ensure that namespaces are unique across PDS, EN creates them in consultation with the discipline nodes. It is critical to avoid collisions among namespaces used in PDS4 labels - the namespace must be chosen from a predefined set. Refer to the *Standards Reference* [2], *Section 6B* or confer with your PDS discipline node to determine and/or select an appropriate namespace for each local dictionary for your archive.

For information on Local Dictionaries, consult Section 11.

PART II. THE FIRST STEPS

4.0 OUTLINE THE BUNDLE

We introduce the 'Voyager 2 Jupiter Encounter Data' archive that was originally produced by the PPI node and delivered to the PDS as a PDS3 dataset. This will provide the basis for all of the specific discussions which follow.

Our spacecraft is the 'Voyager 2' (VG2) spacecraft and the instrument is the 'Plasma Science Experiment' (PLS) a plasma instrument designed to detect plasma conditions throughout the Voyager trajectory.

	Name	***Abbreviation / Acronym***
Spacecraft	Voyager 2	VG2
Instrument	Plasma Science Experiment	PLS

4.1 Collections in the Bundle

Refer to the *Standards Reference [2]*, *Section 2B* and confer with your PDS discipline node to determine all of the required and appropriate collections for your bundles.

Using the archive from our sample data, for the Venus encounter data, the instrument team plans to submit a single "archive" bundle to PDS. The bundle will consist of five collections.

Bundle:

- Browse Collection
- Context Collection
- Data Collection
- Document Collection
- XML Schema Collection

4.2 Directory Organization

PDS has established rules for the organization of data during transfer to, from or within PDS, see the *Standards Reference* [2], *Section 2B.2* for the full set of requirements. Our sample "archive" bundle is a fairly simple bundle that uses a fairly simple directory structure. Since there are only

a small number of collections, we elect to have one directory in the bundle root for each collection.

The bundle root must contain at least one file, the XML label file for the bundle product, and may only contain one additional file – an optional readme file which if used is described in the bundle XML label file.

```
bundle root
| - bundle.xml
|
| - browse
|    | - collection_browse.xml
|    | - collection_browse_inventory.tab
|    | - collection product1
|
| - context
|    | - collection_context.xml
|    | - collection_context_inventory.tab
|
|    | - context_product1
|    | - context_product2
|    | - context_product3
|
| - data
|    | - collection_data.xml
|    | - collection_data_inventory.tab
|    |
|    | - data product1
|
| - document
|    | - collection_document.xml
|    | - collection_document_inventory.tab
|    |
|    | - doc_product1
|    | - doc_product2
|    | - doc_product3
|
| - xml_schema
|    | - collection_xml_schema.xml
|    | - collection_xml_schema_inventory.tab
```

```
|   |
|   | - xml_schema_product1
|   | - xml_schema_product2
|   | - xml_schema_product3
```

The root level subdirectories each correspond to a single collection. Each directory will contain the collection XML label file and the collection inventory file. Depending on the size of the collection, it may or may not contain other files.

In our example data, there is a single observational data file in the data directory, ELEMON.TAB.

However, had there been multiple observational data files, it would have been reasonable for the team to decide to use the UTC date-time or the spacecraft clock count at the start of each observation as the primary reference for each observation. This would have been used as the filename root and in the Logical Identifier (LID) for each observational data product. There are several approaches to setting names for each observational product and the corresponding subdirectories.

4.3 Determine the Documentation Needed

Refer to the *Standards Reference* [2], *Section 8* and confer with your PDS discipline node to determine all of the required and appropriate documentation for your document collection(s).

PDS requires that products, collections, and bundles be documented so that scientists in future years can understand (1) how the data were collected and processed, (2) what the data mean, and (3) the limitations of the data.

Documentation takes three forms in PDS: (a) XML labels, (b) documents included within the archive, and (c) references to material that is publicly available elsewhere. Data providers should negotiate with the PDS consulting node early in the design process on the documentation requirements and how they will be distributed among the three categories above. Documentation considered essential to understanding or using the archive and the underlying data in the archive, except for published journal articles, must be submitted as part of the archive. Journal articles may be included if permitted by the copyright holder. Each document to be archived must be prepared and saved in a PDS-compliant format, Refer to *Section 8A of the Standards* [2] for a list of PDS approved formats.

In our archive example, documentation includes the following documents:

1. An errata file that describes any changes or known errors in the archive.

2. A copy of a published journal article that describes the mission (in both ASCII and HTML).
3. A copy of a published journal article that describes the instrument (in both ASCII and HTML).
4. A checksum file that lists the MD5 checksum of the files in the archive. Note that this file is not a required PDS4 document. It is included in the PDS4 archive simply because it was part of the original PDS3 data_set.

Each of the above document products is individually labeled. Both the errata and the checksum files were each labeled using the Product_File_Text schema as these documents are strictly ASCII text. The other two document products, the mission and instrument descriptions, were each labeled using the Product_Document schema as these documents are presented in both an ASCII and an HTML version. Note that the two forms of the document, the ASCII and HTML versions, are collectively a single document product (i.e., All versions of a document are considered part of a single PDS document product).

For information on how to populate the attributes and classes in a Product_Document, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

5.0 DESIGN LOGICAL AND VERSION IDENTIFIERS

Every product label contains an identifier which must be unique across all products archived with the PDS. This identifier is referred to as a LIDVID and is the concatenation of a Logical Identifier (LID) and a version identifier (VID). We'll address the construction of each in the following sections.

5.1 General Concepts

Here are some general rules:

- LIDs must be unique across PDS
- Each PDS4 LID is constructed as a Uniform Resource Name (URN)
- Each LID in a PDS archive begins with 'urn:nasa:pds'
- LIDs are case insensitive.
- LIDs are restricted to lower-case letters, digits, dash, period, and underscore. Colons are also used but only in a prescribed way to delimit "fields"; discussed in this section.
- Each PDS4 LID is constructed as four (bundle), five (collection), or six (product) fields, where each field is delimited by a colon.
- LID maximum length is 255 characters.

The complete set of requirements for LID construction is given in *Section 6D.2 of the PDS4 Standards Reference* [2].

5.2 Constructing LIDs

Detailed requirements and formation rules are provided in the *Standards Reference* [2], *Section 6D.2*; we provide a brief summary here.

Recall that each basic product is delivered to PDS as a member of a collection, and that collection is a member of a bundle. LIDs are constructed based on a hierarchical set of relationships.

We can think of LIDs as constructed by concatenating fields of characters. The fields are separated by colons. This is the only use of colons permitted in LIDs.

- Bundle LIDs -- are constructed by appending a bundle specific field to 'urn:nasa:pds'.

  Bundle LID = urn:nasa:pds:<bundle field>

  Since all PDS bundle LIDs are constructed this way, the bundle field must be unique across all products archived with the PDS.

- Collection LIDs -- are constructed by appending a collection field to the parent bundle's LID.

Collection LID = urn:nasa:pds:<bundle field>:<collection field>

Since the collection LID is based on the bundle LID, which is unique across PDS, the only additional condition is that the collection field must be unique across the bundle.

- Basic Product LIDs -- are constructed by appending a product field to the parent collection's LID.

  Product LID =
  urn:nasa:pds:<bundle field>:<collection field>:<product field>

  Since the product LID is based on the collection LID, which is unique across PDS, the only additional condition is that the product field must be unique across the collection.

5.2.1 Examples

The following examples are based on a hypothetical mission.

	Name	*abbreviation*
spacecraft	Super SpaceCraft 01	ssc01
instrument	High Resolution Photon Counter	hirespc
cruise phase	Cruise, Mercury, Earth phase	cruise

The team decides to use the spacecraft clock count at the start of each observation as the product field of the LID for observational data products.

This is all the information we need to start designing LIDs.

Cruise Phase
Bundle
urn:nasa:pds:ssc01. hirespc.cruise

Note that in the above example bundle field, ssc01. hirespc.cruise, we used periods as separators. Alternatively we could have used dashes, underscores, or some combination of the three. Discuss the use of period, dash, and underscore in LIDs with your consulting node to determine if the node has a preference.

Collections
urn:nasa:pds:ssc01.hirespc.cruise:browse
urn:nasa:pds:ssc01. hirespc.cruise:context

urn:nasa:pds:ssc01. hirespc.cruise:data
urn:nasa:pds:ssc01. hirespc.cruise:document
urn:nasa:pds:ssc01. hirespc.cruise:xml_schema

Note in the above example we have a single data collection which as you will see below contains both raw and derived data. An alternative organization would be to separate this into two collections, data_raw and data_derived. Either organization is permitted; you should use whichever is best suited for your data. Again, early discussions with your consulting node are strongly encouraged.

Products [data products for sclock = 31234567]
urn:nasa:pds:ssc01. hirespc.cruise:browse:browse_31234567
urn:nasa:pds:ssc01. hirespc.cruise:data:data_raw_31234567
urn:nasa:pds:ssc01. hirespc.cruise:data:data_derived_31234567
urn:nasa:pds:ssc01. hirespc.cruise:document:errata
urn:nasa:pds:ssc01. hirespc.cruise:xml_schema:table_character_0411f

5.3 VID Construction

Detailed requirements and formation rules are provided in the *Standards Reference* [2], *Section 6D.3*; we provide a brief summary here.

Version IDs are used for all types of products, including basic products, collections, and bundles.

- Version IDs (VIDs) are separated from LIDs by a double colon ("::").
- VIDs must be of the form M.n where "M" and "n" are both integers. "M" is the "major" component of the version and "n" is the "minor" component of the version.
- The major number (M) is initialized to '1' for archive products, but the number '0' may be used for sample products or test run products that are not yet ready for the archive. Whenever the major number (M) is incremented, the minor number (n) is reset to '0'.
- Neither M nor n should be prepended with zeros; each is simply incremented as an integer. Thus "1.1" and "1.10" are different versions, and "1.01" is invalid.

The VIDs in all of the products in our sample archive (e.g., bundle, collection, and product) are '1.0' which, if our sample was a science archive instead of an example archive, would correspond to the first submission intended for registration within the PDS.

5.4 LIDVID Construction

A *version identifier* (VID) may be appended to a logical identifier (LID) to identify one of several

versions of the same bundle, collection, or product. The combination is called a *versioned identifier* (LIDVID). LIDVIDs are used to locate products within PDS; every version of every product within PDS has a unique LIDVID.

Concatenate the LID and VID using a double colon as the connector. Here are sample LIDVIDs based the example LIDs in Section 5.2.1

5.4.1 Examples

Note that in the examples LIDs that follow, we used periods as separators. Alternatively we could have used dashes, underscores, or some combination of the three. Discuss the use of period, dash, and underscore in LIDs with your consulting node to determine if the node has a preference.

Cruise Phase

Bundle

urn:nasa:pds:ssc01. hirespc.cruise::1.0

Collections

urn:nasa:pds:ssc01. hirespc.cruise:browse::1.0
urn:nasa:pds:ssc01. hirespc.cruise:context::1.0
urn:nasa:pds:ssc01. hirespc.cruise:data::1.0
urn:nasa:pds:ssc01. hirespc.cruise:document::1.0
urn:nasa:pds:ssc01. hirespc.cruise:xml_schema::1.0

Products [data products for sclock = 31234567]

urn:nasa:pds:ssc01. hirespc.cruise:browse:browse_31234567::1.0
urn:nasa:pds:ssc01. hirespc.cruise:data:data_raw_31234567::1.0
urn:nasa:pds:ssc01. hirespc.cruise:data:data_derived_31234567::1.0
urn:nasa:pds:ssc01. hirespc.cruise:document:errata::1.0
urn:nasa:pds:ssc01. hirespc.cruise:xml_schema:table_character_0411f::1.0

5.5 LIDVIDs – The Next Step

LIDs and LIDVIDs will be ubiquitous in your archive. Be sure you have the naming convention / formation rule / algorithm correct before proceeding. When you have constructed draft LIDs and LIDVIDs contact your PDS DN to verify they are unique and are conformant.

Once you and your DN have settled on a naming convention / formation rule for applying LIDs and LIDVIDs to the various products in your archive, the next step is to apply the formation rule to the pipeline that automatically generates the labels in your archive.

PART III. BASIC PRODUCT LABELS

6.0 BASIC PRODUCT LABELS - AN OVERVIEW

The labels for basic products (all products except Collection and Bundle products) structurally fall into two groups: observational products and all other basic products. We start with the former; then we will discuss the aspects of non-observational product labels which differ slightly. We then discuss the aspects of aggregate products labels and their construction:

- Observational Product Labels
- Non-Observational Product Labels
- Aggregate Product Labels

As a data provider, you should coordinate early with your consulting Discipline Node (DN).

All product label files are XML files generated and validated against a specific version of the "master-schema" maintained by PDS. The data provider and a representative from the DN discuss the anticipated nature of the observational data. Once the DN has sufficient information, the DN data engineer will identify the appropriate products (e.g., Product_Observational, Product_Document, Product_Collection, etc) and the associated schemas (e.g., "master-schema and Schematron" and any locally defined schema and Schematron). The set of schemas / Schematron define the products in your archive and provide validation criteria for ensuring the integrity of the products in your archive. The DN will then pass the set of schemas / Schematron to you, the data provider. The DN may also provide a set of XML label templates; if not those can be generated from the master-schema using an XML aware editor as described later in this chapter.

Each PDS4 label should identify the type of product the label is describing. As discussed later, the various "Product" classes provide the set of options from which you choose the XML label's 'root element'. The options are:

- **Product Browse** - a basic product containing a low resolution or "quick-look" version of an observational product.
- **Product Bundle** - an aggregate product used to identify the member collections of an archive bundle.
- **Product Collection** - an aggregate product used to identify the member basic products of an archive collection.
- **Product Context** - a basic product identifying the physical (instrument, spacecraft, target, people) and conceptual (investigation, node) objects related to an observational product's provenance.
- **Product Document** - a basic product identifying a single logical document, such as an interface specification, instrument description, or user's manual; the document product may comprise multiple formats.

- **Product File Text** - a basic product consisting of a single digital file with ASCII character encoding.
- **Product Observational** - a basic product comprising images, tables, and other fundamental data structures that are the result of a science or engineering investigation.
- **Product SPICE Kernel** - a basic product consisting of a SPICE kernel.
- **Product Thumbnail** - a basic product consisting of a highly reduced version of an observational product, typically used in displaying the results from search interfaces.
- **Product Update** - a basic product containing information about updates to observational products that have already been archived.
- **Product XML Schema** - a product consisting of XML formatted schemas, Schematron files, OASIS catalog files, or any other reference schemas used in the interpretation of an observational product

Note that the class, Ingest_DD, used to generate a local data dictionary (see Chapter 11) is also a permitted option for 'root element'. It is the only class other than the Product_* classes which may be used for 'root element'.

6.1 Basic Product Labels – Observational Products

6.1.1 Selecting the Appropriate Type of Observational Products

Based on the anticipated nature of the observational data, the DN and data provider will negotiate on the types of observational products (e.g., Array, Table, Header, etc) to be included in the data collection(s) of your archive. For observational products, the Product_Observational class in the master-schema is used to define the available product types.

Within the generic Product_Observational class, the available product types are further refined:

For binary array data:

- Array_2D
- Array_2D_Image
- Array_2D_Map
- Array_2D_Spectrum
- Array_3D
- Array_3D_Image
- Array_3D_Movie
- Array_3D_Spectrum

For binary tabular data:

- Table_Binary

For character tabular data:

- Table_Character
- Table_Delimited

For "other" data:

- Header
- Encoded_Header
- Encoded_Binary
- Encoded_Byte_Stream
- Encoded_Image
- Parsable_Byte_Stream
- Stream_Text

Any basic product label may support multiple objects and multiple object types. For example, the Product_Observational class provides the capability to include the description of a Header, an Array_2D_Image, and a Table_Character histogram of pixel values. The three objects, which may or may not all reside in the same file, coupled with the single XML label (that describes the objects) form the single digital product.

The hooks to incorporating additional allowed object types is via the File_Area_Observational class which permits the inclusion of additional digital objects (e.g., Header, Table_Character, etc) that are specified in connection with the Array_2D_Image.

If your data also has browse objects, you may either create separate browse products and associate them with the appropriate observational products using the Reference_Area, or incorporate them in the observational products using the File_Area_Observational_Supplemental. For additional information and advice, contact your consulting DN.

6.1.2 Basic Product Label Organization – Observational Products

In a typical basic product label there are a set number of major blocks of information (areas). Each of these areas contains one or more classes each of which may have several attributes. The areas are:

- XML Preamble
  - Provides a declaration that the document is an XML version 1.0 document
  - Provides the file names and locations of the schema and Schematron files against which the label is validated.

- Root declaration
  - Provides a declaration of the root element of the XML document. The root element is based on the product type. See Section 6.1.1 for a list of the available product types.

    - Identifies the namespaces used in the label (i.e., the 'pds' namespace plus any additional discipline or mission name spaces) including the associated schema.

- Identification Area
    - Provides identification information specific to the product.
    - Provides citation information specific to the product
    - Provides modification history specific to the product

- Observation Area
    - Provides classes describing the specific observation, laboratory experiment, instrument, etc.
    - Includes subsections for relevant classes specific to one or more discipline nodes, and to the mission (or equivalent name space).

- Reference List Area
    - Provides identification information for products, journal articles, etc., relevant to understanding the product. References are made to sources both internal and external to PDS.

- File Area
    - Identifies the file(s) containing the data object(s), and
    - Provides classes specific to each data object in a given file (e.g., the description and parameters of each header, table, image).

Figure 6-1 depicts the major blocks of information (areas) in a typical basic product label.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model
href="http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/
dph_example_products/schemas/PDS4_PDS_0310b.sch"?>

<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation= "http://pds.nasa.gov/schema/pds4/pds/v03
     http://pds.jpl.nasa.gov/pds4/schemas/PDS4_PDS_0310b.xsd">

     <Identification_Area>     27 lines
     <Observation_Area>        76 lines
     <Reference_List>          10 lines
     <File_Area_Observational> 41 lines
</Product_Observational>
```

Figure 6-1. Example Class organization for Product_Observational

All of the basic products adhere to the structure identified in Figure 6-1. This structural consistency enables us to look at the construction of the label for a single basic product type in

detail with the knowledge that the same techniques will be applicable across all label types. This structural consistency also simplifies data management and user browsing.

We will use Product_Table_Character for the following discussion.

Note that Figure 6-1only reflects the content of the labels with respect to the major classes / areas and the underlying hierarchy within the areas. There is a significant amount of content that is not depicted including the sub-classes and attributes contained in the classes.

> ***For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a basic product label, consult the PDS4 Data Dictionary* [3a,3b]. *For information on how to populate the attributes and classes, consult the example PDS4 products described in Section 14 - Example PDS4 Products.***

6.2 Basic Product Labels – Non-Observational Products

The Non-Observational products are still products, so the labels for non-observational products are structurally similar to those of observational products. This section highlights the differences.

6.2.1 Selecting the Appropriate Types of Non-Observational Products

Based on the requirements for documenting the ancillary / non-observational data, the DN and data provider will negotiate on the types of non-observational products (e.g., Documents, Context, Calibration, etc) to be included in your archive. For non-observational products, there is a large set of available product types. An example list of available product types includes:

- Product_Browse
- Product_Calibration
- Product_Context
- Product_Document
- Product_Software
- Product_Thumbnail

The process for selecting the appropriate product types for non-observational products is the same as that used for the observational products.

The DN will provide the schemas (e.g., “master schema and master-schematron” and any locally defined schema and schematron) to you, the data provider.

Note that unlike observational products which can support multiple object types, some of the non-observational product labels only support a single object type. For example, the Product_Document product may only describe a single digital document product although that document product may include multiple formats of the same document (e.g., an instrument description provided in both HTML and PDF/A would be a single product).

6.2.2 Basic Product Label Organization – Non-Observational Products

The organization of a non-observation product label is fairly well aligned to that of an observational product label. The PDS XML labels begin with a section of statements known as the XML preamble. This is followed by a section of statements that declare the root element of the PDS XML label. The preamble and root declarations describe the references to the master-schema and the master-Schematron (both of which reside within the "pds" namespace). Here is an example of these statements:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model
href="http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/dph_example_products
/schemas/PDS4_PDS_0310b.sch"?>

<Product_Document xmlns="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation= "http://pds.nasa.gov/schema/pds4/pds/v03
http://pds.jpl.nasa.gov/repository/pds4/schemas/PDS4_PDS_0310b.xsd">
```

For more information about the XML preamble and the root declaration contact your consulting discipline node. Additional information about the XML preamble and root declarations can be found on the SBN PDS4 Migration Wiki.

In a typical non-observational product label there are a set number of major blocks of information (areas). Each of these areas contains one or more classes each of which may have several attributes. The areas that are shared by all of the non-observational products include:

- XML Preamble
  - Provides a declaration that the document is an XML version 1.0 document

- Root declaration
  - Provides a declaration of the root element of the XML document. See Section 6.2.1 for a list of the available product types.

- Identification Area
  - Provides identification information for the product.

- Context Area

    - Provides context information for the product (e.g., information about the investigation and observing system, time coordinates, target identification, etc).

- Reference List Area
    - Provides identification information for products, journal articles, etc., relevant to understanding the product. References are made to sources both internal and external to PDS.

The following is a partial list of areas that are specific to one or more of the non-observational products:

- File Area_Browse
    - Identifies the file(s) containing the data object(s), and
    - Provides classes specific to each data object in a given file (e.g., the description and parameters of each browse product).

- File Area_Encoded_Image
    - Identifies the file(s) containing the thumbnail data object(s), and
    - Provides classes specific to each data object in a given file (e.g., the description and parameters of each thumbnail product).

- Document_Format_Set
    - Identifies the set of data objects that collectively form the document.
    - Provides identification information specific to each type of document format (e.g., html, pdf).

- Document_Description
    - Provides descriptive information about the document being described.

Note that the above only reflects the content of the labels with respect to the major classes / areas and the underlying hierarchy within the areas. There is a significant amount of content that is not depicted including the attributes contained in the classes.

For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a basic product label, consult the PDS4 Data Dictionary* [3a,3b]. *For information on how to populate the attributes and classes of either a Document or Browse product, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

6.3 Aggregate Product Labels

There are two types of aggregate products:

- Product_Bundle
- Product_Collection

Both of these product types are discussed in detail in Sections 8 and 9, respectively.

7.0 LABEL TEMPLATE CREATION / EDITING

The XML label-template, as the name implies, is an xml file that will be used as a template for generating / producing the actual XML label(s) that will be used in your archive.

We can use the tools provided in an XML aware editor (XAE) to generate a label-template from the master-schema (xsd). Unless label-templates are provided by your consulting node, you will need to generate a label-template for each type of product that will be used in your archive (bundle, collection, and each type of basic product).

The following is a list of the types of label-templates that can exist in your archive:

- Product_Browse
- Product_Bundle
- Product_Collection
- Product_Context
- Product_Document
- Product_File_Text
- Product_Observational
- Product_SPICE_Kernel
- Product_Thumbnail
- Product_XML_Schema

7.1 Creating a Label Template

The XML label-template is an xml file which looks much more like the final label than does the schema. The label-template is created from the master-schema (xsd) using your favorite xml-aware editor (XAE). The steps for creating the label-template are as follows:

Steps:

1. Download a copy of the current master-schema (xsd)
2. Open the master-schema in xml-aware editor (XAE)
3. Customize the settings
4. Create label-template (xml)
5. Ensure label-template is valid

For additional information on how to create a label-template can be found in Appendix D – Steps to Create a Label Template.

7.2 Label Template Editing

Editing a label-template can range from:

- A zero-step process where no modifications are made to the classes and attributes defined in the label-template (i.e., all required and optional classes and attributes are applicable to the label instance).
- A one step process where modifications are made to the classes and attributes defined within the "main body" (i.e., where non-locally defined classes and attributes reside) of the label-template (i.e., the classes and attributes defined in the label-template are "adjusted" to fit the requirements of the label instance).
- An additional follow on step where modifications are made to the areas of the label-template where locally-defined classes and attributes are included (if required).

In the examples in this and following sections, we show fragments of XML templates. Values which vary from one label to the next will be represented by placeholders that either you or the pipeline software will replace.

These generally contain dummy values inserted by the XML editor, for example the value 'name1' may appear as <name>name1</name>. In the label-template, editing the "main body" of the label-template is somewhat less involved than that of editing the areas where locally-defined classes/attributes are defined. We start with the former; then we will discuss the aspects of editing the areas where locally-defined classes/attributes are defined.

We will then discuss XML declaration and schema reference which is applicable across all PDS4 product labels.

7.2.1 Label Editing – Main Body of the Label Template

In terms of making changes / editing the classes and attributes defined in the "main body" of the label template, there aren't that many types of modifications that can be made:

1. You can remove, from the label template, the classes or attributes designated as optional in the parent schema (e.g., the "master-schema" and any locally-defined schema).
2. You can ensure an optional class and/or attribute is present in the label template
3. You can modify the number of times a class or attribute is repeated
4. You can set the value of an attribute to be fixed / static
5. You can specify a value from an enumerated list of values

Keep in mind that all changes / modifications to the label template must adhere to the constraints dictated by the referenced version of master-schema and master-Schematron plus any additional discipline specific schema / Schematron .

We will address each of these types of modifications in the sections that follow.

7.2.1.1 Label Editing - Presence / Absence of a Class or Attribute

Recall that in a schema, either the "master-schema" or any locally-defined schema, the presence and / or absence of a class or attribute is indicated by the minOccurs and maxOccurs values appropriately (i.e., if "minOccurs=0" and "maxOccurs=1", the XML element is not required to be present; but may be present once and only once in the label template).

You have the same control over the presence and absence of classes and attributes within the label template. Use either your favorite text editor or XAE to edit the label template and simply add or delete the class or attribute from the label template. Note that using an XAE allows you to validate in real time as you make your edits.

7.2.1.2 Label Editing – Ensure the Presence of a Class or Attribute

There may be a case where you want to ensure that an optional class or attribute is always included in the label.
In order to ensure that an optional class or attribute appears in your XML label, you will need to write a 'rule' in your XML Schematron file. Strong validation is the key to an accurate archive, so you almost certainly will need to create a mission or discipline specific Schematron file. Your consulting DN may provide one or more discipline specific Schematron files to use in addition to the master-schema and Schematron file. The Schematron file(s) may include constraints such as the Schematron "rule" that follows:

```
<sch:pattern>
 <sch:rule context="//pds:Citation_Information">
  <sch:assert test="pds:description">
     The description in Citation_Information must exist.</sch:assert>
 </sch:rule>
</sch:pattern>
```

The above rule ensures that the description attribute, which is defined in the master-schema as optional, will exist in the label-template and the product labels that are generated by the label-template.

See Appendix B for more information on how to construct Schematron "rules" or consult your discipline node.

7.2.1.3 Label Editing – Repeating a Class or Attribute

In a schema, either the "master-schema" or any locally-defined schema, the number of times a class or attribute will repeat is indicated by setting the minOccurs and maxOccurs values appropriately. In the master-schema in most cases the values are either 0, 1, or unbounded. For example, if the schema specifies that there are 5, and always 5, fields in one row of a table (i.e., "minOccurs=5" and "maxOccurs=5"), initially the XML editor which generated the template will have given the attribute fields a dummy value (50 in this example, not 5 which you would expect), and the template will likely have only one instance of the field subclass (XAEs are XML aware, but not especially bright). You will need to change the value for the field attribute and manually replicate the field class such that you have the 5 instances as specified in the schema.

```
<Record_Character>
     <fields>50</fields>
     <record_length unit="byte">50</record_length>
     <Field_Character>
         <name>name1</name>
         <field_number>1</field_number>
         <field_location unit="byte">1</field_location>
         <data_type>data_type0</data_type>
         <field_length unit="byte">50</field_length>
         <field_format>field_format0</field_format>
         <unit>unit1</unit>
         <scaling_factor>0</scaling_factor>
         <value_offset>0</value_offset>
         <description>description11</description>
     </Field_Character>
          .
          . <add 3 more instances here>
          .
     <Field_Character>
         <name>name5</name>
         <field_number>5</field_number>
         <field_location unit="byte">2550</field_location>
         <data_type>data_type0</data_type>
         <field_length unit="byte">50</field_length>
         <field_format>field_format0</field_format>
         <unit>unit1</unit>
         <scaling_factor>0</scaling_factor>
         <value_offset>0</value_offset>
         <description>description11</description>
     </Field_Character>
 </Record_Character>
```

Often the precise number of replications appropriate for a particular label is specified in a discipline or mission level Schematron file. Repetitions of classes and attributes within the label-template must adhere to the restrictions specified in the reference schema and Schematron .

Here is an example of a Schematron "rule" that ensures that exactly 5 fields are present in one record of a table:

```
<sch:pattern>
 <sch:rule context="pds:Record_Character">
  <sch:assert test="count(pds:Field_Character) eq 5">
    There must be 5 and only 5 instances of field_character</sch:assert>
```

```
  </sch:rule>
</sch:pattern>
```

See Appendix B for more information on how to construct Schematron "rules" or consult your discipline node.

7.2.1.4 Label Editing - Set the Value of an Attribute to be Fixed / Static

For attributes with fixed values, like the number of fields in a record of a fixed width table, you can insert the appropriate value directly into the template. However, in order to ensure that the attribute appears in your XML label with the correct value you will need to write a 'rule' in your XML Schematron file. See Appendix B for more information on how to construct "rules" that will ensure the value of an attribute is fixed / static or the value conforms to an enumerated list of values. An example of a Schematron "rule" follows:

```
<sch:pattern>
  <sch:rule context="pds:Identification_Area">
  <!-- ================================================== -->
  <!-- Test: ensure 'information_model_version' value      -->
  <!--       matches expected-value                        -->
  <!-- ================================================== -->
  <sch:assert test="pds:information_model_version='0.3.1.0.b'">
    Identification_Area.information_model_version: does NOT specify
    the current version.
  </sch:assert>
  </sch:rule>
</sch:pattern>
```

The above rule ensures that the information_model_version is fixed to a value of '0.3.1.0.b'.

```
<Identification_Area>
    <logical_identifier>
        urn:nasa:pds:sampleProducts:array2d_image.mex-v1.0
    </logical_identifier>
    <version_id>1.0</version_id>
    <title>MARS
          PATHFINDER LANDER Experiment</title>
    <information_model_version>0.3.1.0.b</information_model_version>
    <product_class>Product_Observational</product_class>
</Identification_Area>
```

An example of a Schematron rule that ensures the value of an attribute matches one of the values from an enumerated list of values follows:

```
<sch:pattern>
  <sch:rule context="pds:Array">
    <sch:assert test="pds:encoding_type = ('Binary', 'Character')">
      The attribute pds:encoding_type must be equal to one of the following
```

```
      values 'Binary', 'Character'.</sch:assert>
  </sch:rule>
</sch:pattern>
```

For additional information regarding the use of Schematron schemas and the construction of Schematron "rules", see Appendix B or consult your discipline node.

For examples, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

7.2.2 Label Editing – Areas where Locally-defined Classes/Attributes are Defined

In terms of making changes / editing the areas where locally-defined classes and attributes are defined, there are only two such areas:

- Mission_Area
- Discipline_Area

The Mission_Area contains classes and attributes that have been defined in one or more locally-defined data dictionaries that are specific to a particular mission or investigation (i.e., the Mars Express mission could create a data dictionary for the sole purpose of defining attributes and classes that describe instrumentation and science data that is particular to the Mars Express mission).

The Discipline_Area contains classes and attributes that have been defined in one or more data dictionaries that are specific to a particular discipline. For example, the Rings node has attributes / classes that are particular to a large number of Rings products. The Rings node may identify a number of instances where it would be convenient to group Rings descriptors into one or more data dictionaries that are specific to the various types of Ring disciplines.

An example of two locally-defined attributes that reside in the Mission_Area follows:

```
<!-- ============================================================ -->
<!-- Reference the attributes that were imported from the         -->
<!-- local dictionary (using the local namespace (dph)            -->
<!-- ============================================================ -->
<Mission_Area>
    <dph:spacecraft_clock_start_count>1246</dph:spacecraft_clock_start_count>
    <dph:spacecraft_clock_stop_count>unknown</dph:spacecraft_clock_stop_count>
</Mission_Area>
<Discipline_Area>
</Discipline_Area>
```

In terms of editing the Mission_Area and/or the Discipline_Area, the constraints are imposed by the underlying locally-defined schemas (XSDs and SCHs). However, you should always confirm that changes you intend to make within the Discipline_Area do not conflict with the intentions of the DN providing those schema. Our goal, 'our' in the sense of PDS and data provider, should be to ensure that end users receive quality data with quality documentation. We do not need to be involved in searching for loopholes in the requirements.

For information on using Local Dictionaries in your product labels, consult Section 11.

7.3 XML Declaration – Preamble and Root Tag

This section addresses how the XML preamble and root tag of an XML document is created. The preamble and root tag are always the beginning of a PDS XML label.

7.3.1 XML Preamble – XML Declaration

```
<?xml-model
href="http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/dph_exam
ple_products/schemas/PDS4_PDS_0310b.sch"
schematypens="http://purl.oclc.org/dsdl/schematron"?>
```

The first line declares the document to be an XML version 1.0 document using the UTF-8 encoding. The XML declaration must be the first line and is static. When you create your label-template, the XML declaration is auto-populated by the XAE.

```
<?xml version="1.0" encoding="UTF-8">
```

If the XML document is compliant with the ASCII character set (i.e., only contains ASCII characters), then the following XML declaration can be used:

```
<?xml version="1.0" encoding="ASCII">
```

Information about this line can be found at:

http://www.w3.org/TR/REC-xml/#sec-rmd

The second line is a declaration of how to locate the master-Schematron validation file. Note this line must be entered manually.

```
<?xml-model href="http http://pds.jpl.nasa.gov/pds4/schema/released/pds/v03/
/PDS4_PDS_0310b.sch"
schematypens="http://purl.oclc.org/dsdl/schematron "?>
```

Additional discipline or node specific Schematron files are referenced using similar notation.

```
<?xml-model
href="http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/dph_exam
ple_products/schemas/PDS4_PDS_0310b.sch"
schematypens="http://purl.oclc.org/dsdl/schematron "?>
<?xml-model
href="http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/dph_exam
ple_products/schemas/PDS4_IMG_Camera.sch"
schematypens="http://purl.oclc.org/dsdl/schematron "?>
```

The namespace for the Schematron validation language is:

http://purl.oclc.org/dsdl/schematron

A consideration when building product labels is that you might not want to reference the physical location of files, but rather their permanent location once your archive is complete and registered with the PDS. An xml catalog file can be used to map the permanent location listed in the label to a working location on your file system. See Appendix C for more information on using XML Catalog Files.

7.3.2 XML Root Element Declaration – Product Type

The XML preamble is immediately followed by a section of statements that identify / declare the root element of the PDS4 XML label – the root element declaration:

```
<Product_Observational
  xmlns="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation= "http://pds.nasa.gov/schema/pds4/pds/v03
     http://pds.jpl.nasa.gov/pds4/schemas/PDS4_PDS_0310b.xsd">
```

Immediately following the root element, as part of the same XML statement, should be a list of namespaces that are used in the label. In the preceding label snippet, the root element is identified as Product_Observational, everything else is necessary to establish the "pds" and "xsi" namespaces.

Note, for example, the xsi:schemaLocation illustrated above. The xsi is an abbreviation for the XML Schema Instance namespace formally designated by the URI:

http://www.w3.org/2001/XMLSchema-instance

The schemaLocation is an XML attribute of the root element and is defined in the xsi namespace. XML documents may have a default namespace. In this case, bare elements and attributes (i.e., those that are not pre-fixed with a namespace abbreviation followed by a colon)

are assumed to belong to the default namespace. All PDS4 labels should have the 'pds' namespace set as the default namespace.

The root element of the PDS4 XML label is also used to declare the 'pds' namespace (in which the PDS classes and attributes are defined) and additionally any other namespaces that need to be referenced for discipline and node dictionaries (in which locally defined classes and attributes are defined).

```
xmlns:pds="http://pds.nasa.gov/pds4/pds/v03"
xmlns:img="http://pds.nasa.gov/pds4/img/v1"
xmlns:mpf="http://pds.nasa.gov/pds4/mission/mpf/v1"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
```

It is generally considered a good practice to repeat the pds namespace, this time with an explicit abbreviation, in case it is needed somewhere in the label or dictionaries to resolve ambiguities. The xsi namespace must also be listed, as well as any other namespaces needed in the label. Consult with your PDS DN to identify which namespaces to use in your labels.

There are two components to identifying the root element. The first leftmost component specifies one of the approved PDS product types (e.g., Product_Document, Product_Collection, etc). The second component references the URI of the component. The URI of the component is derived from the schema that references the PDS product. In our example above:

- product type: Product_Collection
- URI: xmlns="http://pds.nasa.gov/pds4/pds/v03"

Since the Product_Collection is defined in the master-schema, the URI is derived from the master-schema:

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- PDS4 XML/Schema for PDS4_0.3.1.0.b  Fri Sep 14 18:01:16 PDT 2012 -->
  <!-- Generated from the PDS4 Information Model V0.3.1.0.b -->
  <!-- *** This PDS4 product schema is a preliminary deliverable. *** -->
  <!-- *** It is being made available for review and testing. *** -->
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://pds.nasa.gov/pds4/pds/v03"
    targetNamespace="http://pds.nasa.gov/pds4/pds/v03"
    xmlns:pds="http://pds.nasa.gov/pds4/pds/v03"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    version="0.3.1.0">
```

Notice in the above snippet, from the master-schema, that the "xmlns:pds" namespace is defined as a URI:

```
xmlns:pds="http://pds.nasa.gov/pds4/pds/v03"
```

This same URI is carried over to the XML root element declaration of our example Product_Collection XML document and will be used identically for all products in your archive that are defined by the master-schema

```
<Product_Collection xmlns="http://pds.nasa.gov/pds4/pds/v03"
```

7.3.3 XML Root Element Declaration – Schema Instance

The next statement in the root element declaration declares that the document conforms to the underlying schema (in the XML schema instance namespace).

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

When you create your label-template, the XML declaration is auto-populated by the XML-aware editor.

7.3.4 XML Root Element Declaration – Schema Location

The next statement in the root element declaration specifies the location of the XML schema that is referenced by the XML document. Note that there are variants as to how the schema location is specified. We will address the types of specifications.

In all cases, the xsi:schemaLocation attribute of the root element provides hints to a parsing application as to how to locate the schemas used to assess the validity of the label. The xsi:schemaLocation contains pairs of values which map each namespace to the URI for a schema which defines that namespace. Thus, in the example below, the pds namespace is shown as the first value, and the URI for the PDS4 master-schema, which defines all the PDS classes and attributes for that namespace, is listed second. Remember that a URI is an identifier, not a location, although it may (and in this case does) resolve to a real (online accessible) location.

```
xsi:schemaLocation="
http://pds.nasa.gov/pds4/pds/v03
http://pds.nasa.gov/pds4/pds/v03/PDS4_PDS_0310b.xsd">
```

In the following example, there are two components that collectively map the namespace to the URI of the schema that defines the namespace. The first leftmost component is the URI of the schema referenced by the Product_Collection XML document. The second rightmost component is, in this case, the physical location / path to the schema (e.g., D:/schemas/PDS4_PDS_0310b.xsd).

```
xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v03
file:/D:/schemas/PDS4_PDS_0310b.xsd">
```

When you create your label-template, the XML declaration is auto-populated by the XAE. Both the URI and the location /path are derived from the schema. However, you will most likely need to change the location / path as the label-template and schema progress through the development stage.

7.3.4.1 Schema Location – Physical Location

A consideration when building product labels is that you might not want to reference the physical location of files, but rather their permanent location once your archive is complete and registered with the PDS.

There are several methods for specifying the physical location / path of the schema.

- Schema and label-template are co-located:

```
xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v03
file:/PDS4_PDS_0310b.xsd">
```

- Schema is located in a parent directory to label-template:

```
xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v03
file:./PDS4_PDS_0310b.xsd">
```

- Schema is located in the xml_schema child directory to label-template:

```
xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v03
file:/xml_schema/PDS4_PDS_0310b.xsd">
```

7.3.4.2 Schema Location – Online Accessible Location

Another method for referencing the location of the schema is through a reference to a URL that is online accessible.

- Schema is online accessible:

```
xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v03
http://pds.nasa.gov/schemas/PDS4_PDS_0310b.xsd">
```

7.3.4.3 Schema Location – Using XML Catalog

Yet another method for referencing the location of the schema is by using an XML catalog. An XML catalog is a separate file which you identify using the ‘preferences’ menu in your XAE. The XML catalog is a document describing a mapping between external entity references and locally-cached equivalents. This removes the need to include the reference to the location of the local version while allowing local validation without accessing the online file.

For additional information on using an XML catalog to reference the schema, consult Appendix C on XML Catalog Reference.

7.4 XML Root Element Declaration with Local Dictionary

This section addresses how the root element declaration section of an XML label, having a local dictionary, is created. Most of your Observational products will presumably contain references to one or more local dictionaries. Each dictionary will reside in a namespace.

Our next example uses the Product_Observational product where the product is assumed to reference node and/or discipline specific dictionaries.

The root element declarations of XML labels, that reference local dictionaries, require each namespace to be identified. Our next example includes a reference to a single local dictionary in the "dph" namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model
href="http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/dph_exam
ple_products/schemas/PDS4_PDS_0310b.sch"?>

<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:dph="http://pds.nasa.gov/schema/pds4/dph/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v03
http://pds.jpl.nasa.gov/pds4/schemas/PDS4_PDS_0310b.xsd">
```

From the above example, you can see that the "dph" namespace has the following URI:

```
xmlns:dph="http://pds.nasa.gov/schema/pds4/dph/v01"
```

If you have been following the association between the schema and the referenced URI, you will not be surprised that the "dph" local dictionary specifies the URI that is to be used when referencing the "dph" schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://pds.nasa.gov/schema/pds4/dph/v01"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:dph="http://pds.nasa.gov/schema/pds4/dph/v01"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="0.3.1.0.b">
```

Had the XML document referenced multiple local dictionaries, you simply repeat the reference to each namespace identified by each local dictionary.

```
<?xml version="1.0" encoding="UTF-8"?>
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:dph="http://pds.nasa.gov/schema/pds4/dph/v01"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v03
http://pds.jpl.nasa.gov/pds4/schemas/PDS4_PDS_0310b.xsd">
```

In the above example, both the "dph" and the "phxmd" local dictionaries are referenced, each using a namespace that is unique to the particular node or discipline dictionary.

In the next example, the physical location / path of the master-schema, which resides in the pds namespace, and the two local dictionary schema (e.g., "dph" and "phxmd") are specified. Note that in this particular case, all dictionaries are assumed to be online accessible.

```
<?xml version="1.0" encoding="UTF-8"?>
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:dph="http://pds.nasa.gov/schema/pds4/dph/v01"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v03
http://pds.jpl.nasa.gov/pds4/schemas/PDS4_PDS_0310b.xsd
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/dph/v01
http://pds.jpl.nasa.gov/pds4/schemas/DPH_0310b.xsd
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/phxmd/v01
http://pds.jpl.nasa.gov/pds4/schemas/PHXMD_0310b.xsd">
```

7.5 XML Label – The Closing Tag

In order to for the XML label to be well formed, it must end with a closing tag for the class opened at the beginning of the label.

```
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v03"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v03"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/pds/v03
   http://pds.jpl.nasa.gov/pds4/schemas/PDS4_PDS_0310b.xsd">
            .
            .
            .
</Product_Observational>
```

7.6 Next Steps

In any case, you will want to have a fully functional PDS4 compliant label-template in place before proceeding onto the next step of integrating the pieces into the pipeline production of the various products in your archive.

7.7 Example XML Preambles

For additional information regarding the XML preamble and the root element declaration and how they are used within a Product Label, consult your discipline node.

For examples, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

PART IV. COLLECTIONS, BUNDLES, DELIVERY PACKAGES

8.0 COLLECTIONS

The next higher level in the organizational hierarchy of an archive is the Collection — an inventory of member products and an accompanying label. The inventory and label are known as a Collection Product. Basic products of similar type and content are grouped into a collection. Observational data, for example, will be gathered into one or more data collection(s), documents into a document collection, etc.

In consultation with your consulting DN, for each product type, determine how best to group the products, that share common characteristics, into appropriate collections (*e.g.*, only 'quick-look' or 'browse' products are expected to be assigned to the *browse* collection).

To define a collection, the archivist creates a collection inventory, a specific type of character table that lists the logical identifiers (LIDs or LIDVIDs) of all the products that are members of the collection. The archivist then creates a collection label that describes the collection inventory.

A collection label contains a logical identifier (LID) that uniquely identifies the collection and provides the links between products that share common characteristics.

For a delivery to the PDS, the basic products, included in the delivery and listed in the collection inventory, must be physically located either in the same directory as the collection label, or in one or more subdirectories to it.

For information on how to populate the attributes and classes in a collection, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

8.1 Members of a Collection

The members of a collection are designated as being either primary or secondary.

- Every product must be a primary member of one and only one collection.
  - That collection is the one in which the product is registered in the PDS repository.
- Products already registered in the PDS repository may be secondary members of other collections.
  - For example a collection of mosaic products might include the source products for each mosaic as secondary members of the collection.
- Primary members must be identified in the collection inventory using LIDVIDs.

- Secondary members may be identified in the collection inventory using either LIDs or LIDVIDs based on which is more appropriate for that collection and product.

8.2 Collection Inventory

The collection inventory is a two column character table where each row of the table describes one of the member products of the collection.

The first column of the table specifies whether the product is either a primary (P) or secondary (S) member of the collection. The second column of the inventory table specifies either the LID or LIDVID of the product.

Secondary members, although they are listed in the collection inventory, are not required to be physically included in deliveries to PDS.

8.3 Generating and Populating a Collection Label

Using an XML aware editor, the archivist generates a label template using the master-schema and selecting Product_Collection as the root element.

Two areas in a collection product label, Collection_Area and File_Area_Inventory, differ from those described under basic product labels.

In addition, the description in the Citation_Information class in the Identification_Area is required. It should provide a terse description of the contents of the collection suitable to be displayed in a web browser.

8.3.1 Collection Area

The Collection area contains two attributes – description which is optional, and collection_type which is required. The description attribute is used to provide an overall description of the collection, and although it is optional, it should be used whenever possible to briefly describe the nature of the contents of the collection. The value of the collection_type attribute must be one of the following values:

- Browse
- Calibration
- Context
- Data
- Document
- Geometry
- Miscellaneous

- SPICE Kernel
- XML_Schema

8.3.2 File Area_Inventory

Similar in structure to the File_Area_Observational, this area has many more constraints on it, reflecting the requirements for Collection inventory table formatting and content. It is required to be present.

The File_Area_Inventory contains a reference to a single physical file, and an inventory description detailing the structure in the file. An inventory object is a specific form of delimited table, with predefined columns.

In the following example inventory object, we have included the required attributes and a few, but not all, of the optional attributes. Almost all of the values are enumerated. The lone exception is <records> the value of which is of course specific to the inventory table being described.

```
    <Inventory>
        <offset unit="bytes">0</offset>
        <parsing_standard_id>PDS DSV 1</parsing_standard_id>
        <encoding_type>Character</encoding_type>
        <records>?</records>
        <record_delimiter>carriage-return line-feed</record_delimiter>
        <field_delimiter>comma</field_delimiter>
        <Record_Delimited>
            <fields>2</fields>
            <groups>0</groups>
            <Field_Delimited>
                <name>Member_Status</name>
                <field_number>1</field_number>
                <data_type>ASCII_String</data_type>
                <maximum_field_length unit="byte">1</maximum_field_length>
            </Field_Delimited>
            <Field_Delimited>
                <name>LIDVID_LID</name>
                <field_number>2</field_number>
                <data_type>ASCII_LIDVID_LID</data_type>
                <maximum_field_length unit="byte">255</maximum_field_length>
            </Field_Delimited>
        </Record_Delimited>
        <reference_type>inventory_has_member_product</reference_type>
    </Inventory>
</File_Area_Inventory>
```

The figure below shows six entries from a collection inventory, for a calibrated data collection (data-cal). The first three entries indicate primary members, each identified using a LIDVID. The last three entries indicate secondary members, each identified using a LID.

```
P,urn:nasa:pds:dph:data-cal:prod1::1
P,urn:nasa:pds:dph:data-cal:prod2::1
P,urn:nasa:pds:dph:data-cal:prod3::1
S,urn:nasa:pds:dph:data-raw:prod1
```

```
S,urn:nasa:pds:dph:data-raw:prod2
S,urn:nasa:pds:dph:data-raw:prod3
```

For further discussion regarding producing collection products, see the SBN PDS4 Migration Wiki.

For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a Collection label, consult the PDS4 Data Dictionary* [3a,3b]. *For information on how to populate the attributes and classes in a collection, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

9.0 BUNDLES

The product at the highest level in the product hierarchy is referred to as a Bundle. Like collections, bundles consist of a list of references to products; however in this case, the referenced products are collections. PDS does not impose requirements on how bundles are defined except that (1) bundle LIDs must be distinct within the overall holdings of PDS, and (2) each bundle must be approved by a PDS peer review.

A bundle identifies all of the collections in a given archive; while the collections together identify all of the basic products in the bundle. The inventory and label are known as a Bundle Product. Unlike collection products, the bundle inventory is a table contained in the XML label file describing it, not in a separate table file. A bundle product may optionally include a separate "read me" file.

To define a bundle, the archivist creates a Product_Bundle label. A bundle label contains a logical identifier (LID) that uniquely identifies the bundle and provides the links between the collection products in the archive.

For information on how to populate the attributes and classes in a bundle, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

9.1 Generating and Populating a Bundle Label

Using an XML aware editor, the archivist generates a bundle label by setting the root element to Product_Bundle and generating a label template.

You will create a single bundle product for the archive you are preparing with references to the member collections in the archive.

Three areas in a bundle product label differ from those described under basic product labels:

- Bundle_Area
- File_Area_Text
- Bundle_Member_Entry

In addition, the description in the Citation_Information class in the Identification_Area is required. It should provide a terse description of the contents of the bundle suitable to be displayed in a web browser.

9.2 Bundle Area

The Bundle area contains two attributes – bundle_type and description. This is where you, the data provider, will want to adequately describe the bundle and indicate the type of bundle that you are producing. Since you are creating a bundle that describes an archive, you will set bundle_type to "Archive". A non-archive bundle has a bundle_type of "Supplemental".

```
<Bundle>
    <description>
        Some really good description of the archive goes here
    </description>
    <bundle_type>Archive</bundle_type>
</Bundle>
```

9.3 File_Area_Text

The File_Area_Text of a bundle product XML label is used to reference the optional 'readme.txt' file (i.e., the string of bits) being described by the label. This area is required if a 'readme.txt' is included as part of the archive.

The File_Area_Text class in the template looks something like this:

```
<File_Area_Text>
     <File>
         <file_name>README.TXT</file_name>
         <local_identifier>document.README.TXT</local_identifier>
         <creation_date_time>2010-03-12T11:59:04</creation_date_time>
         <file_size                                    unit="byte">22875</file_size>
         <md5_checksum>5ef7af310b99d8189e670830c954a290</md5_checksum>
     </File>
     <Stream_Text>
         <name>readme.txt</name>
         <local_identifier>document.stream_text</local_identifier>
         <offset unit="byte">0</offset>
         <external_standard_id>TEXT</external_standard_id>
         <encoding_type>Character</encoding_type>
         <description>
             A really good description goes here.
         </description>
         <record_delimiter>carriage-return line-feed</record_delimiter>
     </Stream_Text>
</File_Area_Text>
```

Populating entries in the File_Area_Text of the XML label is remarkably straightforward. The File_Area_Text class consists of a number of optional and required attributes that describe the data file:

- The file name attribute, a required attribute, provides the name of the file being described.
- The local identifier attribute, an optional attribute, provides the name of the local object being described; the value must be unique within the XML label.

- The creation date time attribute, an optional attribute, provides the date/time when the file was created, either in YMD or DOY format.
- The file size attribute, an optional attribute, provides the size (in bytes) of the file.
- The records attribute, am optional attribute, provides the number of records in the file.
- The md5 checksum attribute, an optional attribute, provides the md5 checksum of the file.
- The comment attribute, an optional attribute, provides a brief description of the comment.

The next sub-block, the Stream_Text class, contains details of the 'readme.txt' product component being described by the XML label.

```
<Stream_Text>
    <local_identifier>local_identifier1</local_identifier>
    <offset unit="byte">0</offset>
    <parsing_standard_id>ASCII</parsing_standard_id>
    <encoding_type>Character</encoding_type>
    <description>some description goes here</description>
    <record_delimiter> carriage-return line-feed </record_delimiter>
</Stream_Text>
```

1. As the 'readme.txt' file is always ASCII, the encoding type is set to 'Character'.
2. The offset is set to indicate the starting location (in bytes) of the 'readme.txt' object. Populating the offset attribute for the bundle product is very simple. As there is only a single product being referenced, the offset is always set to '0' bytes.
3. The parsing_standard_id attribute is set to indicate the formal name of the standard used for the structure of the 'readme.txt' file. This is typically ASCII.
4. The record_delimiter attribute is set to indicate the delimiter used to separate records / lines in the files. This is required to be 'carriage-return line-feed'.

Note that the 'readme.txt' file referenced by the bundle product label must be co-located with the label for the bundle product.

9.4 Bundle Member Entry

The Members of a bundle are referenced using the Bundle_Member_Entry area. The Bundle_Member_Entry area is repeated for each collection product referenced in the bundle.

```
<Bundle_Member_Entry>
    <lid_reference>urn:nasa:pds:example.dph.samplearchive:browse</lid_reference>
    <member_status>Primary</member_status>
    <reference_type>bundle_has_browse_collection</reference_type>
</Bundle_Member_Entry>
<Bundle_Member_Entry>
    <lid_reference>urn:nasa:pds:example.dph.samplearchive:context</lid_reference>
    <member_status>Primary</member_status>
    <reference_type>bundle_has_context_collection</reference_type>
</Bundle_Member_Entry>
```

In the Bundle_Member_Entry class:

1. You use either the lid_reference or the lidvid_reference attribute. You normally will only specify one or the other In bundles, collections which are primary members may be specified by either a LID or LIDVID depending on which is more appropriate for the particular bundle. Discuss which option to use with your consulting node.

2. The value for the reference_type is dependent up on the type of collection being referenced:

   - Browse - bundle_has_browse_collection
   - Calibration - bundle_has_calibration_collection
   - Context - bundle_has_context_collection
   - Data - bundle_has_data_collection
   - Document - bundle_has_document_collection
   - Geometry - bundle_has_geometry_collection
   - Miscellaneous - bundle_has_member_collection
   - SPICE Kernel - bundle_has_spice_kernel_collection
   - Schema - bundle_has_schema_collection

3. The value for the member_status attribute specifies whether the collection is either a primary (Primary) or secondary (Secondary) member of the bundle.

Note that for collections which are primary members of a bundle, in deliveries to the PDS the collection products must reside in a subdirectory of the directory containing the bundle label.

For further discussion regarding producing bundle products, see the SBN PDS4 Migration Wiki.

For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a Bundle label, consult the PDS4 Data Dictionary [3a, 3b]. For information on how to populate the attributes and classes in a bundle, consult the example PDS4 products described in Section 14 - Example PDS4 Products.

10.0 DELIVERIES

The organizational products of PDS4 (i.e. bundles and collections) are logical, not physical structures. Products transferred to, within, or from PDS, need to be placed into a physical structure. PDS uses "delivery packages" to accomplish such transfers. In many cases it will be convenient to organize delivery packages into a physical structure that parallels the logical structure of the archive (i.e., the bundle product at the root level with subdirectories for collections, etc.). However, alternate organizations are possible (such as flat directory structures for incremental deliveries of accumulating collections). PDS only requires that the sender and receiver agree on the structure in advance.

In addition to the contents of the delivery package (the bundle product, collection product(s), and basic products), delivery packages contain two additional files: a transfer manifest and a checksum manifest (think of a shipping inventory on the outside of a box delivered to your home).

- Transfer manifest (optional at the discretion of the recipient, such as for very small packages). Provides the location within the package of each product. It contains one record with two fields for each product (including the bundle product) in the package:
    - first field – LIDVID,
    - second field – the file specification for the product label associated with the LIDVID. This includes the file name of the product label, and is given relative to the location of the bundle product.

- Checksum manifest (required for all delivery packages). Contains an entry for every file (not product) in the delivery package, and will be in the standard format used by md5deep, and other similar MD5 generation tools. Each line in the file contains:
    - first field – 32 character hexadecimal MD5 checksum value, followed by two spaces
    - second field the file specification relative to the root directory of the delivery for the file associated with the checksum.

Transfers of products from point to point within and external to the PDS can be accomplished using a variety of mechanisms from electronic transfer by email attachment of a handful of files to the transfer of multiple terabytes of data using external hard drives. The actual transfer mechanism should be coordinated with the node with which you are working.

Any transfer, to, from, or within PDS has three parts:

1. The "Package" containing the material being transferred.
2. The "Transfer Manifest" that maps each product's LIDVID to the file_specification_name of the product's label file, and
3. The "Checksum Manifest" that maps the individual MD5 checksums to each file in the materials being transferred.

10.1 The Package

The term “Package” is a logical (as opposed to physical) concept encompassing the material actually being transferred and the two required manifest files. The material being transferred may be “packaged” using one of several options. The options include ZIP, gzip, tar, physical media such as thumb drives, external hard drives, etc. Any such option must be approved by PDS. There is no requirement to use any of these for the transfer of a handful of files, nor is there a requirement to use software such as zip when the entire transfer is being made with a dedicated external hard drive.

The data provider and the receiving node determine the best ‘packaging’ option for the delivery.

Except for the case of the delivery of a handful of files, either as test samples or as replacements for previously delivered files, the files within the transfer package must be organized in a PDS compliant directory structure (see the *PDS Standards Reference [2], Section 2* for additional information.

10.2 The Transfer Manifest

The “Transfer Manifest” is a file provided with each transfer to, from, or within PDS. The manifest is external to the package. It is a manifest for each product in the package. It maps product LIDVIDs to file_specification_names for each product XML label file. Please note, if the transfer package is a zip file, tar file, etc., the manifest must contain entries for every label file in the package once that package is unpacked. The transfer manifest is defined as a two column fixed-width table where each row of the table describes one of the products in the package. The first column gives the LIDVID of each product in the package. The second column gives the file_specification_name of each product in the package. The file_specification_name provides the name and location of the product label file relative to the location of the bundle product.

```
urn:nasa:pds:example.dph.sample:browse:collection::1.0 ./browse/collection.xml
urn:nasa:pds:example.dph.sample:browse:ele_mon::1.0    ./browse/ele_mom_browse.xml
urn:nasa:pds:example.dph.sample:data:collection::1.0   ./data/collection.xml
urn:nasa:pds:example.dph.sample:data:ele_mon::1.0      ./data/ele_mom_data.xml
```

Since the transfer manifest is external to the package, and hence external to the archive, there is no requirement to provide a separate PDS4 XML label.

10.3 The Checksum Manifest

The “Checksum Manifest” is a file provided with each transfer to, from, or within PDS that is external to the package. It is a checksum manifest for each file in the package. Please note, if the transfer package is a zip file, tar file, etc., the manifest must contain entries for every file in the package once that package is unpacked. Currently PDS uses MD5 checksums, so the checksum

manifest is just an MD5 checksum table that consists of two columns of data that is compatible with the output from an MD5 checksum utility.

```
4a9a9081a3561e54c12b7e1f6ad4c194  md5deep-3.0\CHANGES.TXT
deb4923feb034f6471f44073d3f9496e  md5deep-3.0\COPYING.TXT
fc619ce13794aed2f9f362a52b1abc54  md5deep-3.0\hashdeep.exe
0326aeff13c17b7f8f5a0917628cab91  md5deep-3.0\HASHDEEP.TXT
9432a6dc6bf452bdf3f92e19106d0bbe  md5deep-3.0\md5deep.exe
419106c4e9471facb6baf22fa1565643  md5deep-3.0\MD5DEEP.TXT
0b5cd51e2de15f532fe75bcfbcd0b924  md5deep-3.0\sha1deep.exe
f2c3f93de180d7871fe7b2407434e049  md5deep-3.0\sha256deep.exe
81ad20307fdfc959696f31be160db17a  md5deep-3.0\tigerdeep.exe
770b6eb6911ca29c83ee2b17e3866fa0  md5deep-3.0\whirlpooldeep.exe
```

Since the checksum manifest is external to the package, and hence external to the archive, there is no requirement to provide a separate PDS4 XML label.

10.4 Generating and Populating an XML Label for the "Package"

If a PDS node wishes to register and preserve a delivery package, transfer or checksum manifest, the node must generate an XML label for the resulting product using the appropriate Product_DIP or Product_SIP.

> ***For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of a basic product label, consult the PDS4 Data Dictionary [3a,3b].***

PART V. LOCAL DICTIONARIES AND OTHER DOCUMENTATION

11.0 LOCAL DATA DICTIONARY

A Data Dictionary serves several purposes. First, the dictionary serves as a reference manual to users of the PDS (and other planetary data systems) to define the attributes and classes that are used to describe planetary data and meta-data. Second, the dictionary serves as a reference for data producers (and others) to aid in the design and understanding of data descriptions. Third, the dictionary has the overall responsibility of ensuring the attributes and classes used in the data descriptions are used in a standard, consistent, predictable manner to the point where each attribute and class can be managed and used as a resource.

Conceptually, a data dictionary defines the attributes and classes which may be used in PDS4 product labels. Practically speaking, it must contain human-readable definitions as well as the syntax and semantic constraints placed on values of the attribute. For classes, it provides the explicit list of attributes constituting the class, and indicates which are required, optional, and/or repeatable. It might also indicate that one or more sub-classes are allowed (or required).

Every attribute and class that is used in any PDS label must first be defined in a data dictionary. Ultimately, all dictionaries will be integrated into the PDS4 Information Model which will "build" the PDS4 Data Dictionary document and the associated Mission and Discipline "dictionary" schemata.

Data dictionaries are categorized into:

- Mission specific
- Discipline specific

Mission specific data dictionaries are those that are comprised of attributes and classes specific to a particular mission or investigation (i.e., the Mars Express mission could create a data dictionary for the sole purpose of defining attributes and classes that describe instrumentation and science data that is particular to the Mars Express mission). Depending upon preference, the mission may elect to have a single data dictionary (that describes both instrumentation and science data) or multiple dictionaries where one dictionary is specific to instrumentation descriptors and another is specific science data descriptors.

Discipline specific data dictionaries are those that are comprised of attributes and classes specific to a particular discipline. For example, the Rings node has attributes / classes that are particular to a large number of Rings products. The Rings node may identify a number of instances where it would be convenient to group Rings descriptors into one or more data dictionaries that are specific to the various types of Ring disciplines.

A local dictionary must reside within a namespace that is unique across all other locally defined dictionaries. In order to avoid collisions among namespaces used in PDS4 labels, the namespace must be chosen from a predefined set. Refer to the *Standards Reference [2], Section 6B* or confer with your PDS discipline node to determine and/or select an appropriate namespace for each local dictionary for your archive.

11.1 Local Data Dictionaries - The Mission Area

The Mission_Area, a subclass of the Observation_Area, functionally acts as a container for mission specific classes and attributes of which each are defined in a local data dictionary. The set of locally defined attributes and classes must be prefixed with a 'mission' namespace identifier applicable to the respective dictionary. The 'mission' namespace identifier must be unique across all other locally defined dictionaries.

```
<!-- ========================================================= -->
<!-- Reference the attributes that were imported from the      -->
<!-- local dictionary (using the local namespace (dph)         -->
<!-- ========================================================= -->
<Observation_Area>
       .
       .
       .
  <Mission_Area>
     <dph:spacecraft_clock_start_count>
        1246943630
     </dph:spacecraft_clock_start_count>
     <dph:spacecraft_clock_stop_count>
         unkown
     </dph:spacecraft_clock_stop_count>
  </Mission_Area>
  <Discipline_Area>
  </Discipline_Area>
</Observation_Area>
```

11.2 Local Data Dictionaries - The Discipline Area

The Discipline_Area, a subclass of the Observation_Area, functionally acts as a container for discipline specific classes and attributes of which each are defined in a local data dictionary. The set of locally defined attributes and classes must be prefixed with a 'discipline' namespace identifier applicable to the respective dictionary. The 'discipline' namespace identifier must be unique across all other locally defined dictionaries.

```
<!-- ========================================================= -->
<!-- Reference the attributes that were imported from the      -->
<!-- local dictionary (using the local namespace (dph)         -->
<!-- ========================================================= -->
<Observation_Area>
       .
       .
```

```
        .
    <Mission_Area>
    </Mission_Area>
    <Discipline_Area>
       <img:application_process_id>
          1246943630
       </img:application_process_id>
       <img:application_process_name>
          unkown
       </img:application_process_name>
    </Discipline_Area>
  </Observation_Area>
```

11.3 Building and Using Local Data Dictionaries

Local data dictionaries are categorized into:

- Mission specific
- Discipline specific

This section describes the inter-relationships between the data dictionary schemas, the data dictionary label(s), and the dictionary service that creates the local data dictionary schema.

The dictionary service optionally merges the attributes and classes defined in the local data dictionary with the PDS4 Information Model. This will ensure that your locally defined attributes and/or classes will be contained in the next build of the PDS4 Data Dictionary and the next build of the Generic Mission & Discipline schemata.

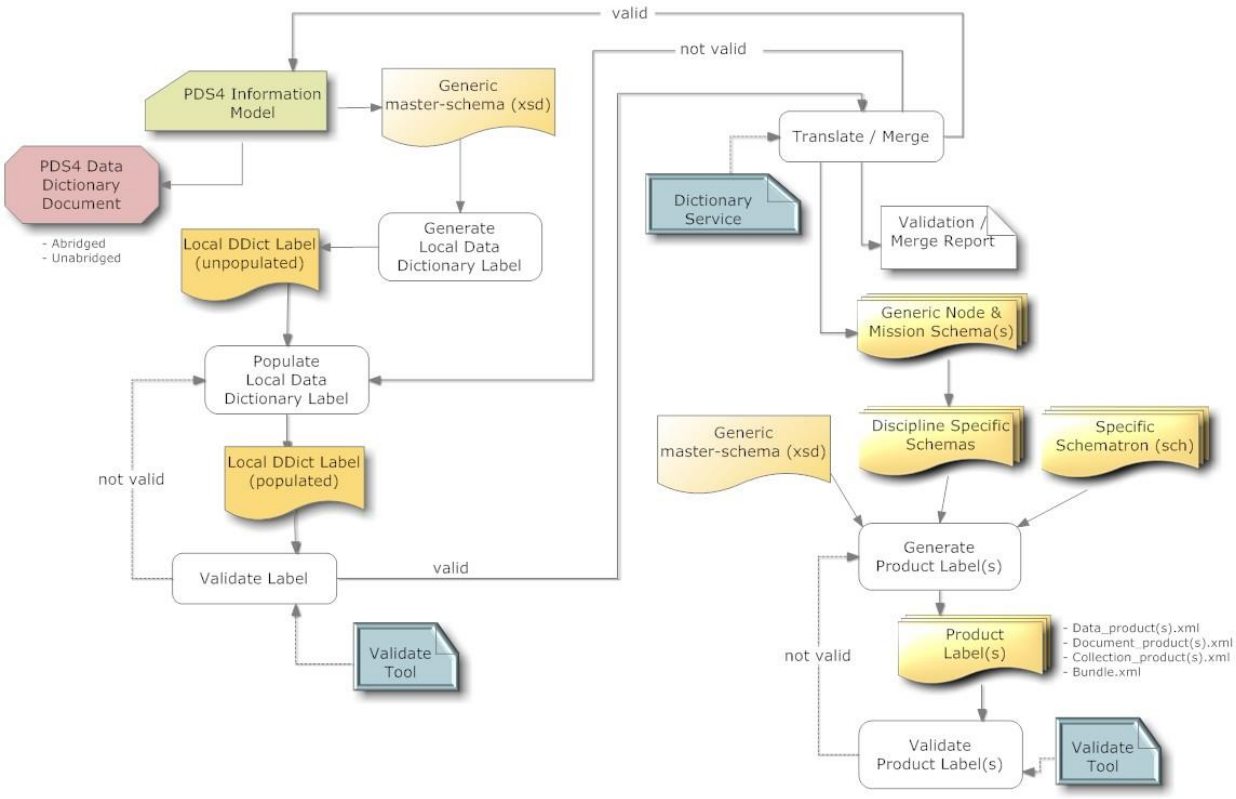


Figure 11-1 Local Data Dictionary Development Flow Diagram

Note that the following examples use Oxygen as the vehicle for demonstrating the local-DD process. In no way does this suggest or imply a recommendation or endorsement for using Oxygen over any of the other XML-aware editors (XAEs).

The steps that follow are a narrative of the major steps to creating a label-template for Ingest_DD. A more detailed description of the generic process of creating a label-template, can be found on the SBN PDS4 Migration Wiki.

Step #1: Download the current version of the master-schema:

http://pds.nasa.gov/pds4/schema/released/pds/

You will want to copy the master-schema (XSD) file to a local directory / folder.

- Open your favorite browser
- Navigate to the above url

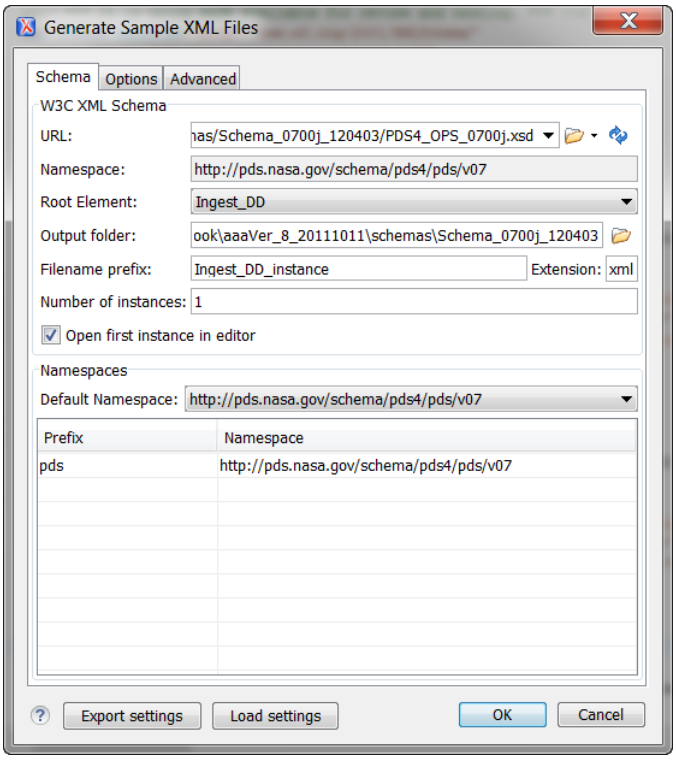
- Select the file of choice to download
- Put your mouse-cursor over the file to be copied
    - right-click and select "Save Link-As" – this will open a dialog box to specify a directory / folder where you want to copy the file

Step #2: Using the XML-aware editor (XAE) of your choice, open the master-schema (XSD) file. Locate the Ingest_LDD class. This is the class from which we will generate the label-template.

Step #3: Most XML-aware editors provide a capability to generate a label-template from a schema. In our case, you will want to generate a label-template from the reference to the Ingest_LDD class.

In Oxygen:

- select: Tools / Generate Sample XML Files.



Notice that the "Schema" tab is active. This tab is used to specify the XSD from which the label-template will be generated.

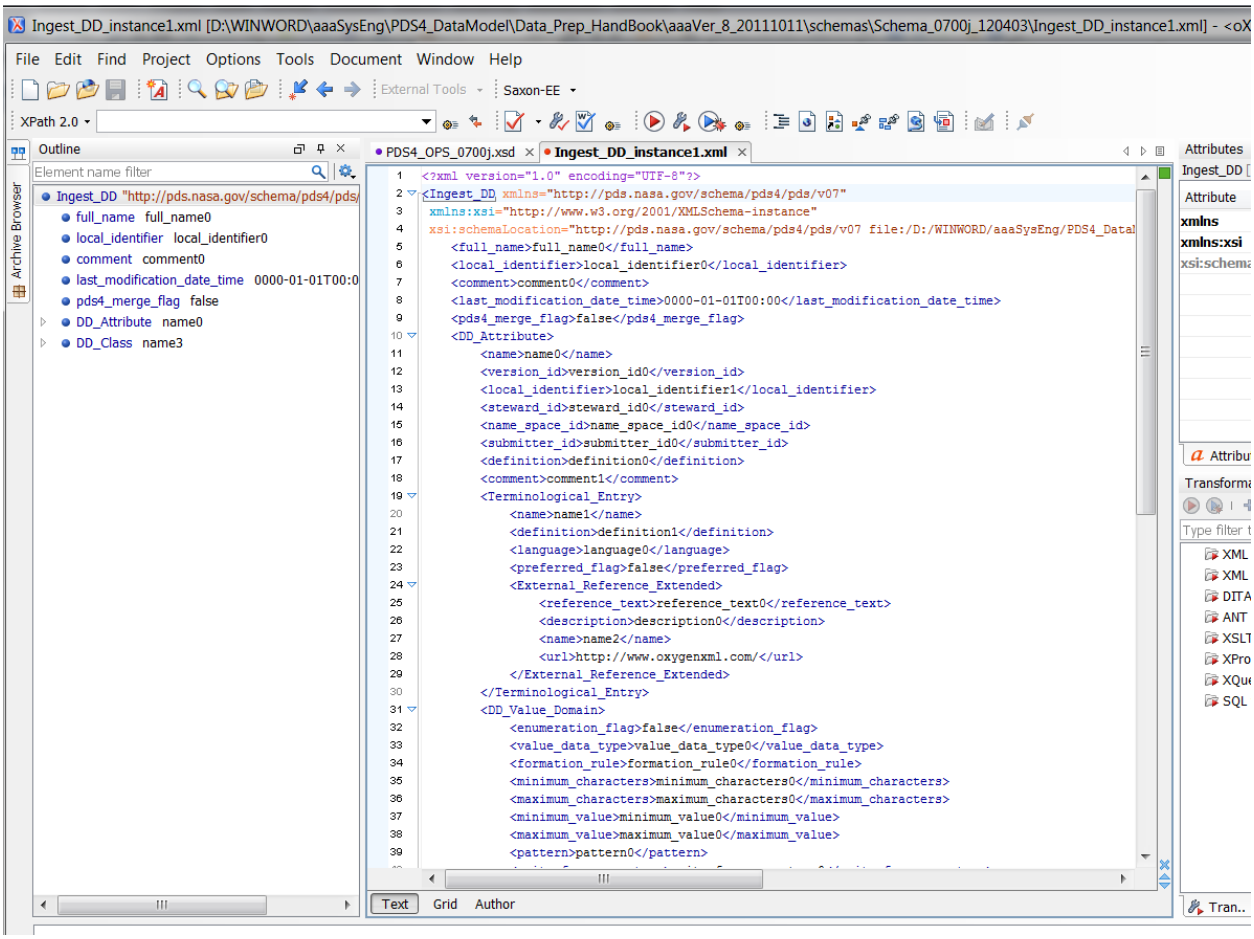
- In URL, you will want to specify the local copy of the master-schema file
- In Namespace, the value should be auto-populated with the value specified in the master-schema. The value should be fairly close to the value displayed above.

- In Root Element, you will want to use the drop-down to select the product for which you want to create a label-template – Ingest_LDD.
- In Output folder, you will want to specify the directory / folder where you want the label-template to reside (when created).
- In Filename prefix, you will want to specify the name of the label-template.
- In Number of instances, you will most likely want a single instance of the label-template to be created. Note that based on the number of instances specified, the value specified in "Filename prefix" will be used as a template for generating multiple instances.
- In Open first instance in editor, you will most likely want to check the box so that the label-template will be displayed in Oxygen (when created).
- In Default Namespace, the value should be auto-populated with the value specified in the master-schema. The value should be fairly close to the value displayed above.
- In Prefix and Namespace, the values should be auto-populated and should be fairly close to the values displayed above.

Once you have verified the above values, select the "Options" tab (at the top of the dialog box).

- In Generate optional elements, you will most likely want to check the box so all optional elements are generated in the label-template.
- In Generate optional attributes, you will most likely want to check the box so all optional attributes are generated in the label-template.
- In Values of elements and attributes, you will most likely want to select "Default" as the option.
- In Preferred number of repetitions, you will most likely want to specify "1" so that a single instance of all elements and attributes will be generated in the label-template.
- In Maximum recursivity level, you will most likely want to specify "1" as the maximum allowed depth in case of recusivity.
- In Choice strategy, you will most likely want to specify "Random" values over "First" value in series.

Once, you verified the above values, click on the "OK" button. This will generate the label-template.



From the above step, you should have successfully created a label-template for the Ingest_LDD product. With the label-template visible, ensure that in the top right, you can see a little green box. If the box is red (or not green), the label-template (xml document) is not valid. As the sample label was generated by the XML editor, there shouldn't be any errors. Contact your PDS rep to resolve any discrepancies.

The label-template when populated will become the data-dictionary label.

Step #4: Examine the as-yet-unedited data dictionary label in your favorite XAE. You are now ready to begin populating the data dictionary label with the metadata associated with each attribute and class that is to be defined in the local data dictionary.

As you populate the dictionary label, ensure that the XML is fully formed (i.e., the XAE will validate the XML and will have an indicator (which is usually a green or red box) that indicates if errors are present in the XML). Or, you can use the PDS4 Validation Tool to further validate the contents of the XML label.

Step #5: At this point, you should have a fully formed (populated) compliant XML data dictionary label. Save all changes to the data dictionary label file.

Step #6: A decision point has been reached whereby the contents of the locally defined data dictionary may be "merged" with the PDS4 Data Dictionary (via ingestion into the PDS4 Information Model (IM). You will need to inform the DE, when you submit the Ingest_LDD.xml for ingestion, of your decision.

Step #7: At this point, you need to send the 'Ingest_LDD.xml" file to the EN, EN will use software to "ingest" the classes and attributes that you defined (in Ingest_LDD.xml) into the IM. EN will then send back to you, the following:

1. A local-dictionary XSD that contains the classes and attributes that you defined.
2. A validation report that indicates either "success" or issues that may need to be addressed.
3. A local-dictionary HTML file that can be browsed for a human-readable dictionary content.

So, the next step is to email the data dictionary label file to the EN for processing.
The EN data engineer will process your file and create a local instance of the "Discipline & Mission" schemata (XSD) and a report describing any anomalies encountered while translating / ingesting.

In the future, you will eventually access the online Data Dictionary Service, "point" the service to your data dictionary label, and press "go". The online Data Dictionary Service is under development.

If you have elected to "merge" the locally defined data dictionary with the PDS4 Data Dictionary, the "service" will additionally validate the contents to ensure a compliant "merge" of the elements and classes is possible (e.g., no duplication of element / class names within the same Registration Authority, etc.). If the EN data engineer doesn't detect any anomalies, then the data engineer will "register" the contents of your data dictionary label with the PDS4 Information Model. This will ensure that your locally defined attributes and/or classes will be contained in the next build of the PDS4 Data Dictionary.

The end result is that you will have online access to the node and mission schemas (XSD).

Step #8: Now that you have a valid local instance of the node and mission schemas (XSD), you can incorporate these dictionaries into your data product pipeline. This is done through an XML include reference in your specific product schema. So, the next step is to "link" the set of discipline and mission schemas (that are to be referenced) into the product schema.

There are four types of references:

- A reference to a schema that is in the same directory as the label-template

- A reference to a schema that is locally-cached / defined
- A reference to a schema that is online accessible
- A reference to a schema using an XML-catalog

In all of the examples that follow, these equivalences are made:

a. the "pds" namespace to: "http://pds.nasa.gov/pds4/pds/v08"
b. the "phxmd" namespace to: "http://pds.nasa.gov/schema/pds4/phxmd/v01"
c. the "phxmd" schema file: "phxmd.xsd"

1. Here is an example of a reference where the schema and the label are in the same directory.

```
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/phxmd/v01 file:/phxmd.xsd">
```

2. Here is an example of a reference where the schema is locally defined.

```
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/phxmd/v01
  file:/D:/schemas/phxmd.xsd">
```

3. Here is an example of a reference where the schema is online accessible.

```
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/phxmd/v01
  http://pds.nasa.gov/schemas/phxmd.xsd">
```

4. Here is an example of a reference where the schema is referenced using an XML-catalog. For more information on using XML-catalogs, consult Appendix C on XML Catalog references.

```
<Product_Observational xmlns="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:pds="http://pds.nasa.gov/pds4/pds/v08"
  xmlns:phxmd="http://pds.nasa.gov/schema/pds4/phxmd/v01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pds.nasa.gov/schema/pds4/phxmd/v01 junk.xsd">
```

Step #9: Now that you have all of the pieces in place, you can incorporate all of these files into your data product pipeline that will pump out gazillions of PDS compliant labels --- don't forget to validate, and then validate again, and again, and again…

11.4 Ingest_DD – Attribute Definitions

For additional information regarding the definitions and cardinality of the attributes and classes used within Ingest_DD, consult the PDS4 Data Dictionary [3a.3b].

11.5 Example Ingest_LDD Product Label

For additional information regarding the definitions and cardinality of the attributes and classes used within the definition of an Ingest_LDD product label, consult the PDS4 Data Dictionary [3a,3b].

An unpopulated sample Ingest_LDD.xml template is available here and a completed Ingest_LDD for the Imaging Telemetry_Parameters class is available here.

12.0 OTHER DOCUMENTATION

Supplementary or ancillary reference materials are usually included with archive products to improve their short- and long-term utility. These documents augment the internal documentation of the product labels and provide further assistance in understanding the data products and accompanying materials. Typical archive documents include:

- Flight project documents
- Instrument papers
- Science articles
- Software Interface Specifications (SISs)
- Software user manuals

The PDS criteria for inclusion of a document in the archive are:

1. What is necessary for evaluation, understanding, and use of the data?
2. What might be useful beyond the 'necessity' threshold?
3. What is publicly available?
4. How should documentation be divided among labels, internal documents, and external documents?

In general, the PDS seeks to err on the side of completeness. Each document to be archived must be saved in a PDS-compliant format as a PDS product. The Product Document and Product_File_Text classes may be used to describe document products. Document products are delivered as part of a document collection.

A flat, human-readable ASCII text version of each document must be included in the archive. Additional versions may be included in other supported formats at the option of the data producer. ASCII text means the file may contain only the standard, 7-bit printable ASCII character set (see Standards Reference [2], Section 6A.3).

A file is human-readable if it is not encoded and if any special markup tags do not significantly interfere with an average user's ability to read the file. So, for example, simple HTML files and TeX/LaTeX files with relatively little markup embedded in the text are generally considered human-readable and may, therefore, be used to satisfy the above ASCII text version requirement.

Note that the PDS takes the requirement for complete documentation very seriously. Documents that are essential to the understanding of an archive are considered as important as the observational data.

Documents prepared for inclusion in an archive are expected to meet not only the PDS label and format requirements, but also the structural, grammatical and lexical requirements of a refereed journal submission. Documents submitted for archiving which contain spelling errors, poor grammar or illogical organization will be rejected and may ultimately lead to the rejection of the submitted data for lack of adequate documentation.

12.1 Internal and External Documentation

To ensure integrity of the archive, the preferred approach is to have all documentation within the archive; in many cases, however, this is either impractical or impossible. For example, when a copyright holder refuses re-use, the copyrighted material must be referenced through an outside source.

The Reference_List class may be used to reference relevant material that resides either inside or outside PDS; it may be associated with any of over two dozen different product classes including Product_Context, Product_Documentation, Product_Collection, and Product_Browse.

The Internal_Reference class may be used to establish relationships with material inside PDS using a LID or LIDVID either through Reference_List or directly, such as from Observing_System_Component.

```
<Internal_Reference>
    <lidvid_reference>
      urn:nasa:pds:context:investigation:mission.voyager::1.0
    </lidvid_reference>
    <reference_type>instrument_host_to_investigation</reference_type>
</Internal_Reference>
```

The External_Reference class provides a complete bibliographic citation to a published work outside the archive, optionally including its Digital Object Identifier (DOI). External_Reference may be used through Reference_List or directly, such as from Observing_System_Component. External_References to inaccessible materials may be rejected during peer review.

```
<External_Reference>
    <reference_text>reference.MORRISON1982</reference_text>
     <description>
         Morrison, D., Voyages to Saturn, NASA SP-451, 227 pp.,
         National Aeronautics and Space Administration, Washington, DC, 1982.
     </description>
</External_Reference>
```

PART VI. VALIDATE THE ARCHIVE

13.0 VALIDATION

This section describes the process of validating the object-oriented design and the inherent relationships of and between the master-schema, the master-Schematron, the specific discipline and mission specific schema and Schematron, and the resulting child XML document.

Figure 13-1 illustrates the process by which users ensure the XML documents are compliant to the parent schemas. The validation process guarantees the object-oriented design of the parent-child relationships are preserved throughout the design and implementation stages of preparing XML documents; specifically that:

- The PDS4 product labels validate/are valid against all of the following:
    a. the master-schema
    b. the specific discipline and mission schema; if provided
    c. Schematron file(s); if provided.

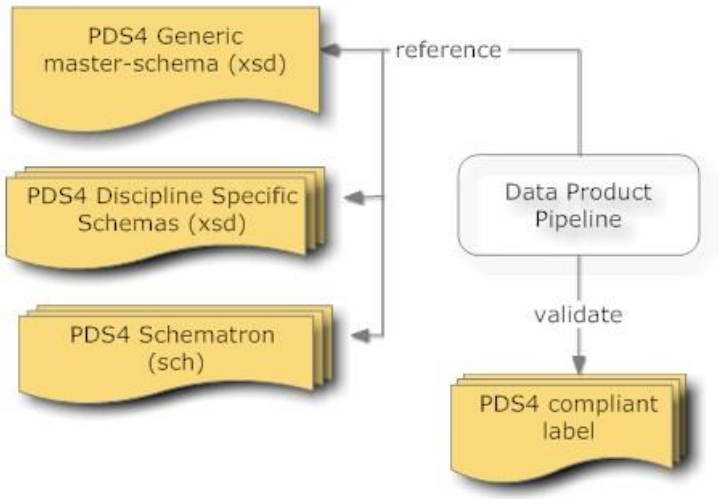


Figure 13-1. Diagram of the Validation of a Product Label

Your Data Product Pipeline will need to incorporate a mechanism by which the pipeline can validate the resulting PDS4 product labels against the parent schema and Schematron files.

13.1 PDS Validation Tool

PDS provides a validate software tool which can be found here:

http://pds.nasa.gov/pds4/software/validate/

PART VI. SAMPLES, EXAMPLES, OTHER TUTORIALS

14.0 EXAMPLE PDS4 PRODUCTS

A PDS4 tutorial would not be complete without providing a set of PDS4 products that were generated from example PDS3 products.

There are two different sets of examples:

- The first is a set of example products. This includes a representative set of products that would exist within an archive (e.g., character table, binary table, document, etc.).

- The second set is a complete archive. This includes all products that would comprise a complete PDS4 archive (e.g., bundle, collections, collection inventories, readme, errata, and basic products). The archive is presented in a directory structure required of an archive.

  This material was originally archived under PDS and is representative of a PDS3 dataset migrated to become a PDS4 "example" archive. Consequently, the products within the "example" archive use "identifiers" that identify the products as "example.DPH" products (i.e., the "identifiers" are not representative of what an actual science archive would use).

  This is intentional and allows these products to be registered with the PDS registry service and searchable as "example products" and "example archive".

14.1 PDS4 Product Examples

The set of PDS4 product examples can be found at:

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/

Within this set of examples, there is an example of the following product types:

1. ARRAY_2D_IMAGE extension of the PDS4 Array_Base, (i.e., Homogeneous N-dimensional array of Scalars) class where a contiguous stream of BINARY data, assembled as a two dimensional data structure, maps the "items" contained in a ARRAY_2D_IMAGE file.

2. TABLE_CHARACTER extension of the PDS4 Table_Base (i.e., Heterogeneous repeating record of Scalars) class where a contiguous stream of ASCII characters, assembled as fixed-width fields, maps the "items" contained in a TABLE_CHARACTER file.

3. TABLE_CHARACTER_GROUPED extension of the PDS4 Table_Base (i.e., Heterogeneous repeating record of Scalars) class where a contiguous stream of ASCII characters, assembled as sets of repeating fixed-width fields, maps the "items" contained in a TABLE_CHARACTER_GROUPED file.

4. TABLE_BINARY extension of the PDS4 Table_Base (i.e., Heterogeneous repeating record of Scalars) class where a contiguous stream of BINARY data, assembled as fixed-width fields, maps the "items" contained in a TABLE_BINARY file.

5. TABLE_BINARY_PACKED extension of the PDS4 Table_Base (i.e., Heterogeneous repeating record of Scalars) class where a contiguous stream of BINARY data, assembled as fixed-width fields, maps “Packed_Data_Fields” and “Field_Bit” classes contained in a TABLE_BINARY file.

6. DELIMITED_TABLE class where a contiguous stream of ASCII characters, combined with a field_delimiter and record_delimiter scheme, maps the "items" contained in a CSV “like” file.

7. DOCUMENT class where one or more instantiations of a document (e.g., ascii text, pdf, html), as identified as a set, comprise a logically complete “copy” of the referenced document product.

8. HEADER and TABLE_CHARACTER classes illustrating how combined digital objects can co-exist in a single data product file.

The files that comprise the PDS4 example products can be found at:

- PDS4 migrated data_set files (directory listing):

  http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/dph_example_products/

A zip file of the PDS4 migrated data can be downloaded from:

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/zip_ExampleProducts.zip

14.2 PDS4 Example Archive

This set of PDS4 examples were derived from a PDS3 PPI data_set:

```
DATA_SET_ID = "VG2-J-PLS-5-SUMM-ELE-MOM-96.0SEC-V1.0"
```

Both the original PDS3 data_set files and the equivalent PDS4 product files are presented for the purposes of illustrating (comparatively) how a PDS3 data_set can be migrated to PDS4.

14.2.1 PDS3 Data_set Files

The directory structure of the original PDS3 PPI data_set consisted of the following:

- aareadme.txt
- checksums.txt
- errata.txt
- \browse
- \catalog
- \data
- \documents
- \documents\mission
- \documents\pls
- \documents\symbols

The files that comprise the original PDS3 PPI data_set can be found at:

- PDS3 PPI data_set files (directory listing):

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/dph_example_PDS3_VG2PLS/

A zip file of the PDS3 VG2PLS data_set can be downloaded from:

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/dph_example_PDS3_VG2PLS/ VG2PLS.zip

14.2.2 PDS4 Product Files

The PDS4 archive structure, as migrated from the above PDS3 files, consists of the following:

- Product_Bundle.xml
- README.TXT
- \browse
- \context
- \data
- \document
- \xml_schema

The following PDS3 files have been relocated to the “document” directory:

- `checksums.txt`
- `errata.txt`

The files that comprise the PDS4 migrated data_set can be found at:

- PDS4 migrated data_set files (directory listing):

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/dph_example_archive_VG2PLS/

A zip file of the PDS4 migrated data can be downloaded from:

http://pds.jpl.nasa.gov/repository/pds4/examples/dph_examples_3b/dph_example_archive_VG2PLS/zip_VG2PLS_archive.zip

APPENDIX A ACRONYMS

The following acronyms pertain to this document:

ADM	Architecture Development Method
API	Application Programming Interface
COTS	Commercial Off-The-Shelf
EN	Engineering Node (PDS)
ESDIS	Earth Science Data and Information System
FTP	File Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IPDA	International Planetary Data Alliance
IT	Information Technology
JPL	Jet Propulsion Laboratory
NASA	National Aeronautics and Space Administration
NSSDC	National Space Science Data Center
PDS	Planetary Data System
RM-ODP	Reference Model of Open Distributed Processing
RSS	Really Simple Syndication
SDSC	San Diego Supercomputing Center
SOA	Service-Oriented Architecture
TB	Terabyte
TOGAF	The Open Group Architecture Framework
XAE	XML aware editor
XML	eXtensible Markup Language

APPENDIX B SCHEMATRON REFERENCES

In markup languages, **Schematron** is a rule-based validation language for making assertions about the presence or absence of patterns in XML trees. It is a structural schema language expressed in XML using a small number of elements and XPath.

Schematron is capable of expressing constraints in ways that other XML schema languages like XML Schema and DTD cannot. For example, it can require that the content of an element be controlled by one of its siblings. Or it can request or require that the root element, regardless of what element that is, must have specific attributes. Schematron can also specify required relationships between multiple XML files.

Constraints and content rules may be associated with "plain-English" validation error messages, allowing translation of numeric Schematron error codes into meaningful user error messages.

An example of a Schematron "rule" follows:

```
    <sch:pattern>
      <sch:rule context="pds:Identification_Area">
      <!-- ================================================= -->
      <!-- Test: ensure 'information_model_version' value      -->
      <!--       matches expected-value                      -->
      <!-- ================================================= -->
      <sch:assert test="pds:information_model_version='0.3.1.0.b'">
        Identification_Area.information_model_version: does NOT specify
        the current version.
      </sch:assert>
   </sch:rule>
</sch:pattern>
```

The above rule ensures the value for the information_model_version attribute in the Identification_Area exactly matches '0.3.1.0.b':

A primer to Schematron is available on the SBN PDS4 Migration Wiki.

- A good set of introductory tutorials are available here. There is no need to run through all of the tutorials. But Two Types of XML Validation, Usage and Features, and Overview will set the stage of what Schematron is :
  - http://www.xfront.com/schematron/index.html

- Great set of starter examples and how to:
  - http://zvon.org/xxl/SchematronTutorial/General/toc.html

- A walk through of Schematron :
  - http://www.ibm.com/developerworks/xml/tutorials/x-schematron/

APPENDIX C XML CATALOG REFERENCES

XML documents typically refer to external entities, for example schemas for the Document Type Definition. These external relationships are expressed using URIs, typically as URLs.

However, if they are absolute URLs, they only work when your network can reach them. Relying on remote resources makes XML processing susceptible to both planned and unplanned network downtime.

Conversely, if they are relative URLs, they're only useful in the context where they were initially created. For example, the URL "../../xml/dtd/docbookx.xml" will usually only be useful in very limited circumstances.

One way to avoid these problems is to use an entity resolver or a URI Resolver. A resolver can examine the URIs of the resources being requested and determine how best to satisfy those requests. The XML catalog is a document describing a mapping between external entity references and locally-cached equivalents.

The following is an example of a XML catalog that resolves the locations of schemas as the schemas follow a somewhat natural development cycle of: DEV, ARCHIVE, and ONLINE.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <!-- ========================================================= -->
  <!-- 1st reference is to DEV instance                          -->
  <!-- ========================================================= -->
    <group xml:base="file:/D:/DEV/schemas/">
    <uri name="http://pds.nasa.gov/pds4/pds/v03" uri="PDS4_PDS_0310b.xsd"/>
    <uri name="http://pds.nasa.gov/schema/pds4/phxmd/v01" uri="PHXMD_0310b.xsd"/>
    <system systemId="PDS4_PDS_0310b.xsd" uri="PDS4_PDS_0310b.xsd"/>
  </group>

  <!-- ========================================================= -->
  <!-- 2nd reference is to ARCHIVE instance                      -->
  <!-- ========================================================= -->
  <group xml:base="file:/D:/ARCHIVE/schemas/">
    <uri name="http://pds.nasa.gov/pds4/pds/v03" uri="PDS4_PDS_0310b.xsd"/>
    <uri name="http://pds.nasa.gov/schema/pds4/phxmd/v01" uri="PHXMD_0310b.xsd"/>
    <system systemId="PDS4_PDS_0310b.xsd" uri="PDS4_PDS_0310b.xsd"/>
  </group>

  <!-- ========================================================= -->
  <!-- 3rd reference is to ONLINE instance                       -->
  <!-- ========================================================= -->
  <system systemId="http://pds.nasa.gov/schema/pds4/phxmd/v01"
uri="http://pds.jpl.nasa.gov/schemas/PHXMD_0310b.xsd"/>
  <uri name="http://pds.nasa.gov/schema/pds4/phxmd/v01"
uri="http://pds.jpl.nasa.gov/schemas/PHXMD_0310b.xsd"/>

</catalog>
```

Once the XML catalog has been created, the catalog file will need to be "registered" with the XML-aware editor that you are using. Once registered, you will also need to validate that the XML-catalog is being referenced by your XML product labels.

- A primer to XML Catalogs is available on the SBN PDS4 Migration Wiki.

  The primer will take you through the following steps:

  1. *Example Schema / XML Labels)*
  2. *XML Catalog Solution, Part 1: schemaLocation Changes*
  3. *XML Catalog Solution, Part 2: Creating an XML Catalog File*
  4. *XML Catalog Solution, Part 3: Configuring the XML Catalog*
  5. *XML Catalog Solution, Part 4: Validating the XML document using the XML Catalog*
  6. *XML Catalog Solution, Part 5: Using XML Catalog to make schemas Portable*

APPENDIX D STEPS TO CREATE A LABEL-TEMPLATE

This appendix illustrates the steps that can be used to create a label-template from the master-schema (xsd).

Steps:

1. Download a copy of the current master-schema (xsd)
2. Open the master-schema in an xml-aware editor (XAE)
3. Customize the settings
4. Create label-template (xml)
5. Ensure label-template is valid

Step 1: Download a copy of the current master-schema (xsd)

The current version of the PDS4 master-schema (xsd) can always be found here:

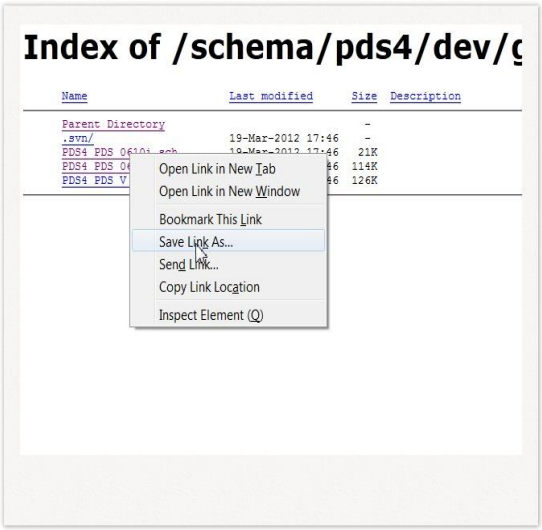
http://pds.nasa.gov/pds4/schema/released/pds/

The two files of interest are:

- The current Schematron file: PDS4_PDS_xxx.sch
- The current master-schema file: PDS4_PDS_xxx.sch

You will want to copy both of the above files to a local directory / folder.

- Open your favorite browser
- Navigate to the above url
- Select the file of choice to download
- Put your mouse-cursor over the file to be copied
    - right-click and select "Save Link-As" – this will open a dialog box to specify a directory / folder where you want to copy the file
    - repeat for the 2nd file

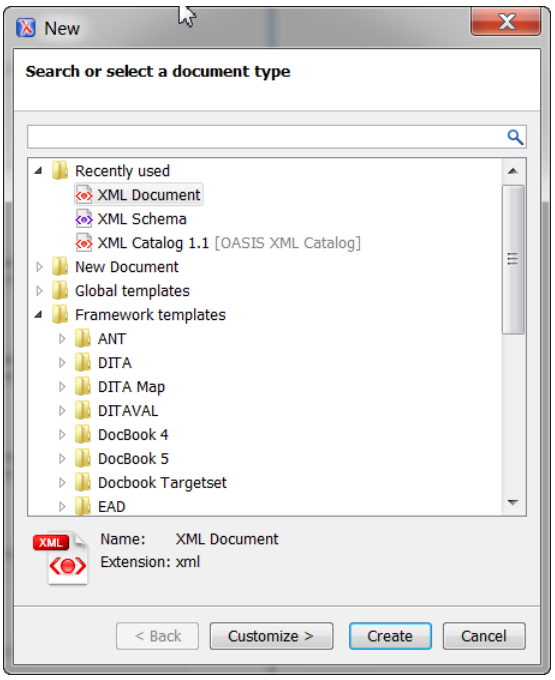


Step 2: Open the master-schema in XML-aware Editor

From Step 1, you should have a local copy of the master-schema (xsd) and the Schematron (SCH) files.

Using your favorite xml-aware editor (XAE), open the master-schema (xsd) file.

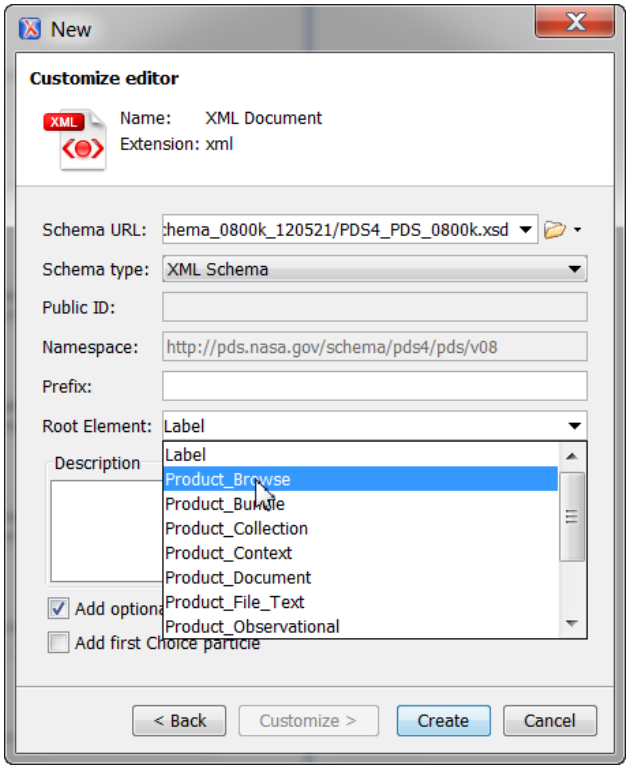
The next step is to identify the product for which you want to create a label-template. The XAE The next step is to the XAE to create a new document. You will want to create a new "XML Document".



Step 3: Customize the Settings

Before you create the document you we will need to "customize" some of the information that will be needed to create the label-template (xml) document that will suit your purpose.

The master-schema includes references to a number of "products" each of which represents a distinct set of product classes. Product classes are, by definition, the highest level classification of the product into a category like "observational product" or "document":



1. We need to populate the 'Schema URL" with the full path to the master-schema. Either drag-and-drop the master-schema (xsd) or type the path.
2. For 'Schema type', select 'XML Schema' as the type of schema being referenced.
3. The 'Namespace' value should be pre-populated with the namespace identified in the master-schema.
4. For 'Root Element' select the type of product that you want to create (e.g., Product_Browse, Product_Observational, etc).
5. You will most likely want optional content to be added when the label-template is created, so select 'Add optional content'.

Step 4: Create the label-template (xml) document

You are now ready to have the XAE create the label-template:

1. Click on ‘Create’.
2. The label-template will auto-display in the XAE.
3. You will probably want to ‘Save’ the file locally.

Step 5: Ensure the label-template is valid (or not)

From Step 4, you should have a local copy of the label-template (xml) document. The XAE provides a simple visual indication of whether or not the XML file is valid.

The label-template that we generated from Step 4 will most likely not be valid as there are no values specified within the xml tags. Once a full set of values has been specified, the label-template should become valid:

1. With the label-template visible in the XAE, ensure that in the top right, you can see a little green box. If the box is red (or not green), then label-template (xml document) is not valid.